86/04/24

CDC ADVANCED COMMUNICATIONS

# PROGRAM INTERFACE HANDBOOK
# DISTRIBUTED COMMUNICATIONS NETWORK SOFTWARE

Built off of level 1602

Table of Contents

ABORT SYSTEM

```
{ PROCEDURE NAME:  abort_system
{
{ PURPOSE:
{   Bring the system to a halt.
{
{ CALL FORMAT:
{   (*callc CSXABRT)
{   abort_system (halt_code, message_ptr);
{
{ DESCRIPTION:
{   The caller of abort_system passes in a pointer to an adaptable string
{ containing message text about why the abort was necessary.  The halt_code
{ is a more general indication of the area that brought the system down.
{ Using the DI Resident Debugger routine dird_output, the message text will
{ be sent to the screen of the terminal attached to the DI (if present).
{ If the reset code is within the range for a valid reset code
{ (see deck:  SIDRC), then a call is made to reset_di with the specified
{ halt_code, otherwise a call is made to dead_stop where a default
{ halt_code is used.
{
{ CALLS:
{   dird_output
{   dead_stop
{   reset_di
{

  PROCEDURE [XDCL] abort_system ({
                        halt_code: integer;
                        message_ptr: ↑string (* <= dbc$single_line));
```

ABORT TASK

```
{ PROCEDURE NAME:  abort_task
{
{ PURPOSE:
{   Abort Task.
{
{ CALL FORMAT:
{   (*callc CMXMTSK)
{   abort_task (abort_code, task_id, status);
{
{ DESCRIPTION:
{   The indicated task is checked to see if it has a parent
{   task.  If it does not, it is stopped with Stop Task; this
{   effectively brings the entire system to a screeching halt.
{   If it does, the task is suspended and the parent task is
{   notified with an intertask message.
{
{   This is intended to be a response to an illogical software
{   condition, invoking action from the parent to recover or
{   restart the aborted task.
{
{ NOTES:
{   Refer to Executive ERS section 4.22.
{

  PROCEDURE [INLINE] abort_task ( {
        abort_code: integer;
        task: task_ptr;
    VAR status: boolean);
```

ABS, MAX, MIN

```
{ FUNCTION NAME: abs, max, min
{
{ PURPOSE:
{   Numeric Functions.
{
{ CALL FORMAT:
{   (*callc CMXPMMA)
{   value := abs (a);
{   value := max (a, b);
{   value := min (a, b);
{
{ DESCRIPTION:
{   These functions are quite predictable.
{   The parameters to these functions must be numeric; however,
{   the size is immaterial, as the compiler will convert them
{   if that is necessary.

  FUNCTION [XDCL] abs (a: integer): integer;
```

APPEND

```
{ PROCEDURE NAME: append
{
{ PURPOSE:
{   Append Trailer to Message.
{
{ CALL FORMAT:
{   (*callc CMXPAPP)
{   append (size_of_trailer,addr_of_trailer,message_pointer,
{           threshold,allocation_type,result_status);
{
{ DESCRIPTION:
{   The message is checked for use by multiple data streams.
{   If any portion is so used, that portion is logically copied.
{
{   A long buffer(data buffer) is obtained
{   from the executive at the specified threshold.
{
{   The trailer is then copied into the buffer, and the buffer
{   is attached at the end of the message (via a call to Assemble).
{   The Count in Message field of the descriptor is maintained.
{
{   In the event that a copy operation is required, the copied
{   buffers will be released before returning to the caller.
{
{ NOTES:
{   Note that, due to the logical CSMCOPY operation, message returned
{   may be different than message supplied.
{
{ SEE ALSO:
{   Assemble, Trim, Prefix, Logical Copy
{

  PROCEDURE [XDCL] append ( {
        size_of_trailer: non_empty_message_size;
        addr_of_trailer: ↑cell;
    VAR msg_pointer: buf_ptr;
        threshold: threshold_size;
        allocation_type: pref_type; { conditional/unconditional call
    VAR success: boolean);
```

ASCII CHARACTER DEFINITIONS

```
{ TABLE NAME:  ASCII character definitions
{
{ DECK NAME:  CMDASCI
{

  CONST
    nul = CHR (00(16)),
    soh = CHR (01(16)),
    stx = CHR (02(16)),
    etx = CHR (03(16)),
    eot = CHR (04(16)),
    enq = CHR (05(16)),
    ack = CHR (06(16)),
    bel = CHR (07(16)),
    bs = CHR (08(16)),
    ht = CHR (09(16)),
    lf = CHR (0a(16)),
    vt = CHR (0b(16)),
    ff = CHR (0c(16)),
    cr = CHR (0d(16)),
    so = CHR (0e(16)),
    si = CHR (0f(16)),
    dle = CHR (10(16)),
    dc1 = CHR (11(16)),
    dc2 = CHR (12(16)),
    dc3 = CHR (13(16)),
    dc4 = CHR (14(16)),
    nak = CHR (15(16)),
    syn = CHR (16(16)),
    etb = CHR (17(16)),
    can = CHR (18(16)),
    em = CHR (19(16)),
    sub = CHR (1a(16)),
    esc = CHR (1b(16)),
    fs = CHR (1c(16)),
    gs = CHR (1d(16)),
    rs = CHR (1e(16)),
    us = CHR (1f(16)),
    sp = CHR (20(16)),
    del = CHR (7f(16));
```

ASSEMBLE

```
{ PROCEDURE NAME: assemble
{
{ PURPOSE:
{   Assemble Message Fragments.
{
{ CALL FORMAT:
{   (*callc CMXPASS)
{   assemble (fragment_1, fragment_2, threshold);
{
{ DESCRIPTION:
{   The message "fragment_1" is searched for multiple use. If it
{   is multiply used, the portion so used is logically copied and
{   released.
{
{   "fragment_2" is then attached to the tail of "fragment_1"
{   by moving the pointer to the trailing descriptor of "fragment_1."
{
{ NOTES:
{   If the first buffer of fragment_1 is multiply used, new_message
{   will be different than fragment_1.
{
{ SEE ALSO:
{   Fragment
{

  PROCEDURE [XDCL] assemble ( {
      VAR fragment_1: buf_ptr, {address of first message fragment
          fragment_2: buf_ptr; {address of second fragment
          threshold: threshold_size); {threshold for buffer acquisition
```

BROADCAST

```
{ PROCEDURE NAME:  broadcast
{
{ PURPOSE:
{   Prepare Message for Broadcast.
{
{ CALL FORMAT:
{   (*callc CMIPBRO)
{   broadcast (message, destination_count+self);
{
{ DESCRIPTION:
{   The user count of the first descriptor of the message
{   is updated to show the increased number of data streams
{   which must release the message before it may be physically
{   released.
{
{   In the event that a copy operation is required, the copied
{   buffers will be released before returning to the caller.
{
{   The parameter self must be used in the event that the caller
{   wishes to maintain a copy of the message for his own use.
{

  PROCEDURE [INLINE] broadcast ( {
    VAR message: buf_ptr;
        number_of_new_data_streams: 1 .. 32767);

    IF message <> NIL THEN
      message↑.usage_descriptor := message↑.usage_descriptor +
            number_of_new_data_streams;
    IFEND;

  PROCEND broadcast;
```

BUFFER

```
{ TABLE NAME:  buffer
{
{ PURPOSE:
{   Buffer Constants and Types.
{
{ CALL FORMAT:
{   (*callc CMDTBUF)
{    VAR
{      name: buf_ptr;
{
{ DESCRIPTION:
{   This definition describes a descriptor, which is actually a
{   small buffer.  Small chunks of data may be allocated in it;
{   larger ones are in a large buffer (naturally).  These
{   buffers are linked by the "next_descriptor" field; the next
{   chain will be linked by "next_message".  These things are
{   also known as "messages" or just "buffers".
{
{ CAUTION:
{   The user data field of the data_descriptor record should be
{   used with caution. The programmer must be aware of what the
{   common subroutines do with buffers; for example, buffers(s)
{   may be released when strip is called.
{
{   WARNING - There are three options defined by the TYPE pref_type.
{             These options are used in calls to APPEND , PREFIX , and
{             BUILD_HEADER_IN_PLACE. Internally the options have the
{             following meaning when the routine obtains data buffers.
{
{             absolute@    - Use the sure ( TRAP 1 ) interface ; always
{                            return a successful status.
{             conditional@ - Use the maybe ( TRAP 0 ) interface ; return
{                            the status from the EXEC to the user.
{             yield@       - Use the maybe ( TRAP 0 ) interface ; if
{                            successful, return that status to the user.
{                            If not yield, and repeat the process.
{
{             The absolute@ and yield@ options both potentially give up the
{             CPU. If a non-preemptible task issues a TRAP 0 request that
{             fails ( in this case a data buffer request ), the EXEC will
{             execute a CHK instruction. This is why some software will
{             use the yield@ option. However, there are many cases where
{             the yield@ option is inappropriate. Examples include
{             command processors and layer software that executes
{             as directly-called subroutines under other tasks.
{
{             Refer to the EXEC ERS for a more detailed explanation of
{             CPU Scheduling.
{

  CONST
?IF ccdbg THEN
```

```
    max_buffer_size = 128,
?ELSE
    max_buffer_size = 2304,
?IFEND
    max_chars_in_buffer = max_buffer_size - 2,
    critical_priority = 0,
    default_sbufflen = 32,
    default_lbufflen = 144,
    high_priority = 1,
    max_sbufflen = 64,
    max_lbufflen = max_buffer_size,
    medium_priority = 2,
    min_sbufflen = 32,
    min_lbufflen = 64,
    low_priority = 3,
    memory_overhead = 6,
    lbuff_overhead = memory_overhead+2,
    self = 1; { added to destination_count for broadcast, etc.

  TYPE
    non_empty_message_size = 1 .. 65535,
    message_size = 0 .. 65535,
    chars_in_buffer = 0 .. max_chars_in_buffer,
    non_empty_buffer = 1 .. max_chars_in_buffer,
    pref_type = (absolute@, conditional@, yield@); { See WARNING above

  TYPE
    data_descriptor = record
      next_descriptor: ↑data_descriptor, { next buffer in msg
      next_message: ↑data_descriptor, { next msg in queue
      the_data: ↑data_space_record, { the good stuff's here
      decstamp: integer, { millisecond time stamp
      offset: non_empty_buffer, { distance from the_data to 1st byte
      count_buffer: chars_in_buffer, { # bytes data in buffer
      count_message: message_size, { # bytes data in message (1st buffer only)
      usage_descriptor: 0 .. 32767, { usage count of descriptor
      user_data: data_descriptor_user_data_type, { user defined data
    recend,

    data_space_record = record
      data_usage: 0 .. 32767, {usage count for data space
  ?IF  ccdbg  THEN              { DI version follows
                                { Note the "+1". Because the
                                { usage count is only one cell on CYBER
      data_text: ARRAY [1..max_chars_in_buffer+1] OF CELL,
  ?ELSE                                          { CC debugger version
      data_text: STRING (max_chars_in_buffer),
  ?IFEND
    recend,
    buffer_request_limit = 1 .. 999,

    executive_extent = 1 .. 32750, { size of executive extent
                                   { NOTE:  If executive_extent changes MEMMAX
                                   {        in EXDEQUA also needs to be changed.
```

```
    buffer = ↑data_descriptor, { archaic; for C compatibility on
    buf_ptr = ↑data_descriptor;

{
{   The following are definitions of user defined data kept in the
{   data_descriptor record. These fields are normally unused since
{   the common subroutines request sbufflen ( 32 ) bytes for the
{   data_descriptor record.
{

  TYPE
    data_descriptor_user_data_type = record
      case integer of

      = 1 = { XEROX TRANSPORT
        sequence: 0 .. 0ffff(16),

      = 2 = { TDSM (text_processor)
        text_process_1: ↑cell,
        text_process_2: ↑cell,

      = 3 = { TDSM (output_queue)
        marked_output: boolean,

      casend,
    recend;
```

BUILD HEADER IN PLACE

```
{ PROCEDURE NAME: build_header_in_place
{
{ PURPOSE:
{   Build Space for Header on Message.
{
{ CALL FORMAT:
{   (*callc CMXPBLD)
{   build_header_in_place (length, addr, message, threshold, success);
{
{ DESCRIPTION:
{   The subroutine gets a buffer or descriptor as needed,
{   creates space in the message to hold the specified header,
{   and returns both the new message address (via the msgbuf
{   parameter, which is passed by reference), and the address
{   of the header structure. The current first buffer is used
{   if the header will fit and start on an even byte.
{
{   This routine is equivalent to CSMPREF, except that header
{   construction occurs after the call rather than before;
{   it represents a performance upgrade.
{
{ NOTES:
{   The size of a header that is allocated may not exceed
{   the size of the data space of a data buffer; Thus, headers
{   larger than that should be generated using CSMPREF.
{
```

```
  PROCEDURE [XDCL] build_header_in_place ( {
          length: non_empty_buffer; {#SIZE(your typedef)
      VAR addr: ↑cell; {pointer the address of the header.
      VAR message: buf_ptr; {address of a location which
                            {in turn addresses the message.
          threshold: threshold_size; {the buffer allocation threshold.
          allocation_type: pref_type; {absolute or conditional allocation
      VAR success: boolean); { type of return
```

CALL AFTER INTERVAL,

```
{ PROCEDURE NAME:  call_after_interval,
{                  fg_after_interval
{
{ PURPOSE:
{   Call Subroutine with Parameter after Interval.
{
{ CALL FORMAT:
{   (*callc CMXMTIM)
{   call_after_interval (interval, parameter, subroutine, timer_id);
{   fg_after_interval (interval, parameter, subroutine, timer_id);
{
{ DESCRIPTION:
{   The subroutine is called by the Timer Task when the interval
{   requested has expired.
{   Refer to Executive ERS section 4.14.
{
{   The following calls have the following effects:
{
{   NAME:                  TRAP NUMBER:    EFFECTS:
{
{   call_after_interval        0           enqueues timer request
{
{   fg_after_interval          2           for interrupt routines only;
{                                          enqueues timer request
{
{   The function time(hours, minutes, seconds) is also defined
{   in this file to permit time_of_day and interval to be
{   specified in a readable manner.
{
{   E.g., midnight is either time (0, 0, 0) or time (24, 0 , 0).
{   1:53:22 PM is time (13, 53, 22),
{   an interval of 10 seconds is time (0, 0, 10)
{
{ SEE ALSO:
{   Cancel Timer Request
{

  PROCEDURE [INLINE] call_after_interval ( {
        interval: milliseconds;
        parameter: ↑cell;
        timer_routine: ↑procedure (parameter: ↑cell);
    VAR timer_request_identifier: ↑timer);
```

CALL AT TIME,

```
{ PROCEDURE NAME:  call_at_time,
{                  fg_at_time
{
{ PURPOSE:
{   Call Subroutine with Parameter at Time.
{
{ CALL FORMAT:
{   (*callc CMXMTIM)
{   call_at_time (time_of_day, interval, parameter, subroutine, timer_id);
{   fg_at_time (time_of_day, interval, parameter, subroutine, timer_id);
{
{ DESCRIPTION:
{   The subroutine is called by the Timer Task when the time of
{   day has been reached.  If the requested time is prior to the
{   current time (eg, it is now 12:05 am and midnight=0 is
{   requested), the request is understood to expire on the next
{   day.
{   Refer to Executive ERS section 4.12.
{
{   The following calls have the following effects:
{
{   NAME:            TRAP NUMBER:    EFFECTS:
{
{   call_at_time          0          enqueues timer request
{
{   fg_at_time            2          for interrupt routines only;
{                                    enqueues timer request
{
{   The function time(hours, minutes, seconds) is also defined
{   in this file to permit time_of_day and interval to be
{   specified in a readable manner.
{
{   E.g., midnight is either time (0, 0, 0) or time (24, 0 , 0).
{   1:53:22 PM is time (13, 53, 22),
{   an interval of 10 seconds is time (0, 0, 10)
{
{ SEE SLAO:
{   Cancel Timer Request
{

  PROCEDURE [INLINE] call_at_time ( {
        time_of_day: milliseconds;
        parameter: ↑cell;
        timer_routine: ↑procedure (parameter: ↑cell);
    VAR timer_request_identifier: ↑timer);
```

CALL PERIODIC,

```
{ PROCEDURE NAME:  call_periodic,
{                  fg_periodic
{
{ PURPOSE:
{   Call Subroutine with Parameter Periodically.
{
{ CALL FORMAT:
{   (*callc CMXMTIM)
{   call_periodic (time_of_day, interval, parameter, subroutine, timer_id);
{   fg_periodic (time_of_day, interval, parameter, subroutine, timer_id);
{
{ DESCRIPTION:
{   The subroutine is called by the Timer Task when the time of
{   day has been reached.  If the requested time is prior to the
{   current time (eg, it is now 12:05 am and midnight=0 is
{   requested), the request is understood to expire on the next
{   day.
{
{   The call is then repeated at intervals until the request is
{   cancelled, or until the requesting taskid is no longer
{   valid.  Requests from interrupt routines must be actually
{   cancelled.  Refer to Executive ERS section 4.13.
{
{   The following calls have the following effects:
{
{   NAME:            TRAP NUMBER: EFFECTS:
{
{   call_periodic        0        enqueues timer request
{
{   fg_periodic          2        for interrupt routines only;
{                                 enqueues timer request
{
{   The function time(hours, minutes, seconds) is also defined
{   in this file to permit time_of_day and interval to be
{   specified in a readable manner.
{
{   E.g., midnight is either time (0, 0, 0) or time (24, 0 , 0).
{   1:53:22 PM is time (13, 53, 22),
{   an interval of 10 seconds is time (0, 0, 10)
{
{ SEE ALSO:
{   Cancel Timer Request
{

  PROCEDURE [INLINE] call_periodic ( {
        first_expiration: milliseconds;
        interval: milliseconds;
        parameter: ↑cell;
        timer_routine: ↑procedure (parameter: ↑cell);
    VAR timer_request_identifier: ↑timer);
```

CANCEL TIMER,

```
{ PROCEDURE NAME:  cancel_timer,
{                  fg_cancel_timer
{
{ PURPOSE:
{   Cancel Timer Request.
{
{ CALL FORMAT:
{   (*callc CMXMTIM)
{   cancel_timer (timer_id, parameter, status);
{   fg_cancel_timer (timer_id, parameter, status);
{
{ DESCRIPTION:
{   A previously requested timing function is cancelled.
{   Refer to Executive ERS section 4.15.
{
{   The timer_id is returned with a NIL value so that it
{   will not be used again (inadvertantly, of course).
{   A false status is returned if the timer_id is NIL
{   (i. e., if the timer_id is canceled more than once).
{   The DI will be reset if the timer_id is invalid.
{
{   The following calls have the following effects:
{
{   NAME:            TRAP NUMBER:  EFFECTS:
{
{   cancel_timer          0        timer is cancelled
{
{   fg_cancel_timer       2        for interrupt routines only;
{                                  timer is cancelled
{
{ SEE ALSO:
{   Call Subroutine at Time
{   Call Subroutine after Interval
{   Call Subroutine Periodically

  PROCEDURE [INLINE] cancel_timer ( {
    VAR t: ↑timer;
    VAR parameter: ↑cell;
    VAR status: boolean);
```

CHANGE TIMER OWNER

```
{ PROCEDURE NAME:  change_timer_owner
{
{ PURPOSE:
{   Change allocator task id of timer request
{
{ CALL FORMAT:
{   (*callc CMXMTIM)
{   change_timer_owner ( timer_id, task_id, status );
{
{ DESCRIPTION:
{   The requested timing function will have its allocating task id changed
{   to the indicated task. If the indicated task equals nil the current
{   running task will be used as the new allocating task for the indicated
{   timer.
{   A false status is returned if the timer_id is NIL.
{   The DI will be reset if the timer_id or task_id is invalid.
{
{
{   NAME:            TRAP NUMBER:  EFFECTS:
{
{   change_timer_owner   0         allocating task id changed
{
{ SEE ALSO:
{   Call Subroutine at Time
{   Call Subroutine after Interval
{   Call Subroutine Periodically
{   Call Subroutine Cancel_timer
{
  PROCEDURE [INLINE] change_timer_owner ( {
        t: ↑timer;
        task: task_ptr;
    VAR status: boolean);

    PROCEDURE [XREF] call_fast_bg ( index: integer;
                                    t: ↑timer;
                                    task: task_ptr )↑ cell;

    VAR parameter: ↑cell;

    parameter := call_fast_bg ( 5 , t , task );
    status := ( parameter <> NIL );

  PROCEND change_timer_owner;
```

CHECKSUM NEXT MODULE

```
{ PROCEDURE NAME: checksum_next_module
{
{ PURPOSE:
{   Successively validate the section checksums.
{
{ CALL FORMAT:
{   *callc dlxcknm
{   checksum_next_module(load_identifier, next_module_found, checksum_valid);
{
{ DESCRIPTION:
{   The load_identifier parameter is initially passed in as NIL in order to
{ start with the first module.  Successively calculate the checksums for the
{ sections of the current module.  Compare these checksums with the checksums
{ in the module header.  IF they are equal, then checksum_valid is set to
{ true, otherwise, it is returned false.  When the last module was
{ checksummed then, the parameter, next_module_found is returned with
{ a value of false.  The module use count of the previous module is
{ decremented and the current module is incremented.
{

  PROCEDURE [XDCL] checksum_next_module
    ({
     VAR load_identifier: dlt$load_id_ptr;
     VAR next_module_found: boolean;
     VAR checksum_valid: boolean);
```

CLEAR ALLOCATE

```
{ PROCEDURE NAME:  clear_allocate
{
{ PURPOSE:
{   Allocate memory from the system heap, and clear it (to zeros)
{
{ DESCRIPTION:
{   This procedures waits for the allocated memory, and clears the
{   obtained memory.
{
{ NOTES:
{   The allocated memory will always be an even number of bytes,
{   and start at an even byte boundary.
{

  PROCEDURE [XDCL, #GATE] clear_allocate ( {
        memory_bytes: 1 .. 32766) {:} ^cell;
```

CLEAR MEMORY

```
{ PROCEDURE NAME:  clear_memory
{
{ PURPOSE:
{   Clear a given number of memory bytes.
{
{ DESCRIPTION:
{   This procedures clear a given number of memory bytes.
{
{ NOTES:
{   It is assumed that the memory starts at an even byte address,
{   and is of an even number fo bytes.
{

  PROCEDURE [XDCL, #GATE] clear_memory ( {
        even_start_address: ↑cell,
        memory_bytes: 0 .. 32766);
```

CLEAR WRITE PROTECT

```
{ PROCEDURE NAME: clear_write_protect
{
{ PURPOSE:
{   Clear the write protect flag
{
{ CALL FORMAT:
{   (*callc cmicwp)
{   clear_write_protect;
{
{ SEE ALSO:
{   set_write_protect
{
{ NOTE:
{   The proper use of this routine is in conjunction with set_write_protect
{   The order of use should be:
{     clear_write_protect;
{     <modify the normally write protected area of memory>
{     set_write_protect;
{


  PROCEDURE [INLINE] clear_write_protect;
    ptr_control_commands↑.clear_write_protect := 0;
  PROCEND clear_write_protect;
```

CLOSE INTERNET SAP

```
{ PROCEDURE NAME:  close_internet_sap
{
{ PURPOSE:
{   Closes a SAP for an INTERNET user.
{
{ CALL FORMAT:
{   (*callc b3xreqi)
{   close_internet_sap (sap_id, user_id, return_code);
{
{ DESCRIPTION:
{   Find_sap_entry is called to locate the corresponding SAP table entry.
{   If the supplied user ID corresponds to the SAP table entry, the
{   corresponding SAP table entry is released and a new internet SAP
{   table built with the index to the released entry removed.
{
{ GLOBAL INPUT:
{   none
{
{ GLOBAL OUTPUT:
{   open_ephemeral_sap_count - number of ephemeral SAPs open
{   internet_sap_table - pointer to SAP table
{$

  PROCEDURE [#GATE, XDCL] close_internet_sap (
        sap_id: sap_id_type; { INPUT - SAP ID of SAP to close
        user_id: ↑cell; { INPUT - user identifier
    VAR return_code: close_internet_sap_status); { OUTPUT
```

CLOSE STATUS SAP

```
{ PROCEDURE:  close_status_sap
{
{ PURPOSE:
{   The  purpose  of this procedure is to allow a  software
{   component to close a previously opened status sap.
{
{ CALL FORMAT:
{       (*callc sdxssar)
{       close_status_sap (sap_number)
{
{ DESCRIPTION:
{   A     software    component    directly    calls    the
{   close_status_sap  routine  to close a previously opened
{   sap.
{
{   Parameter Description
{   sap_number: (input)
{       This  parameter  uniquely identifies the sap previously
{       opened.   The  sap_number   must   be   the  sap_number
{       returned on the open_status_sap call.
{
{ GLOBAL DATA REFERENCED:
{   software_status_sap_table
{
{ GLOBAL DATA MODIFIED:
{   software_status_sap_table
{
{ NOTES AND CAUTIONS:
{   The    procedure   NOPREMPT   is   called   upon   entering
{   close_status_sap    to     suppress     task    preemption.
{   Close_status_sap is exited in a non-preemptable  state  and
{   will require the caller  to  make  a  call to the procedure
{   OKPREMPT if preemptability is so desired.


  PROCEDURE [XDCL] close_status_sap ( {
        sap_number: software_sap_range);
```

CLOSE 3A SAP

```
{  PROCEDURE NAME:  close_3a_sap
{
{  PURPOSE:
{    This procedure is provided by Intranet to allow users to close an
{    Intranet SAP via a direct call.
{
{  DESCRIPTION:
{    A user of Intranet calls the close_3a_sap procedure directly.  The user
{    must provide the SAP identifier returned on the open_3a_sap request issue.
{
{    If the sap specified is out of range or not active then an error
{    is returned to the user of Intranet via the close_status parameter
{    and the error is logged.
{
{  RETURNS:
{    Name          Type              Description
{    close_status  13a_status_type   This parameter indicates the status of the
{                                    close_3a_sap request.
{
{  GLOBAL DATA REFERENCED:
{    sap_table

  PROCEDURE close_3a_sap ( {
        sap: intranet_sap_type;
    VAR close_status: 13a_status_type);
```

CLP CONVERT INTEGER TO STRING

```
{ Procedure Name:  clp_convert_integer_to_string
{
{   The purpose of this request is to convert an integer to its string
{ representation in a specified radix. The result is left justified in the
{ returned string.  If the integer is negative, the first character of the
{ result is a minus sign (-). If the specified radix is greater than ten and
{ the leftmost digit of the result is greater than nine, then a leading zero
{ digit is added to the result.
{
{ Call Format:
{       (*callc clxci2s)
{       CLP_CONVERT_INTEGER_TO_STRING (INT, RADIX, INCLUDE_RADIX_SPECIFIER,
{         STR, STATUS)
{
{ INT: (input) This parameter specifies the integer to be converted.
{
{ RADIX:  (input)  This  parameter  specifies the radix in which the integer's
{       value is to be represented.
{
{ INCLUDE_RADIX_SPECIFIER:  (input)  This  parameter  specifies  whether  the
{       representation  of the radix is to be included in the resulting string
{       -- e.g.  (16) for a number with a radix of 16.
{
{ STR: (output) This parameter specifies  the  string  representation  of  the
{       integer.
{
{ STATUS: (output) This parameter specifies the request status.

  PROCEDURE [XDCL, #GATE] clp_convert_integer_to_string ALIAS 'clpci2s' ( {
        int: integer;
        radix: 2 .. 16;
        include_radix_specifier: boolean;
    VAR str: ost$string;
    VAR status: clt$status);
```

CLP CONVERT STRING TO INTEGER

```
{ Procedure Name:  clp_convert_string_to_integer
{
{   The purpose  of this request is to convert the string representation of an
{ integer to an integer.  The string representation may contain a leading sign
{ (+ or -) and/or a trailing radix enclosed in parentheses.
{
{ Call Format:
{       (*callc clxcs2i)
{       CLP_CONVERT_STRING_TO_INTEGER (STR, INT, STATUS)
{
{ STR: (input) This parameter specifies the string to be converted.
{
{ INT: (output)  This  parameter  specifies  the converted integer value along
{       with the radix in which the integer was represented.
{
{ STATUS: (output) This parameter specifies the request status.

  PROCEDURE [XDCL, #GATE] clp_convert_string_to_integer ALIAS 'clpcs2i' ( {
    str: string ( * );
    VAR int: clt$integer;
    VAR status: clt$status);
```

CLP CONVERT TO RJSTRING

```
{ Procedure Name:  clp_convert_to_rjstring
{
{   The purpose of this request is to convert an integer to its string
{ representation in a specified radix. The result is right justified in the
{ returned string.  If the integer is negative, a minus sign (-) is included
{ in the result either just to the left of the converted integer if the fill
{ character is a space, or as the leftmost character of the result string. If
{ the specified radix is greater than ten and the leftmost digit of the result
{ is greater than nine, then a leading zero digit is added to the result if
{ the result string is long enough to hold it.
{
{ Call Format:
{       (*callc clxcirs)
{       CLP_CONVERT_INTEGER_TO_RJSTRING (INT, RADIX, INCLUDE_RADIX_SPECIFIER,
{         FILL_CHARACTER, STR, STATUS)
{
{ INT: (input) This parameter specifies the integer to be converted.
{
{ RADIX:  (input)  This  parameter  specifies the radix in which the integer's
{       value is to be represented.
{
{ INCLUDE_RADIX_SPECIFIER:  (input)  This  parameter  specifies  whether  the
{       representation  of the radix is to be included in the resulting string
{       -- e.g.  (16) for a number with a radix of 16.
{
{ FILL_CHARACTER: (input) This parameter specifies the character used to  fill
{       unused positions in the returned string.
{
{ STR: (output)  This  parameter  specifies  the  string representation of the
{       integer.
{
{ STATUS: (output) This parameter specifies the request status.

  PROCEDURE [XDCL, #GATE] clp_convert_to_rjstring ALIAS 'clpcirs' ( {
        int: integer;
        radix: 2 .. 16;
        include_radix_specifier: boolean;
        fill_character: char;
    VAR str: string ( * );
    VAR status: clt$status);
```

CLP GET PARAMETER

```
{ Procedure Name:  clp_get_parameter
{
{   The purpose  of  this  request  is to return the entire value list for the
{ specified parameter, in  its  uninterpreted  form,  as  a  string.   If  the
{ requested parameter was not given, a null string is returned.
{
{ Call Format:
{       (*callc clxgpar)
{       CLP_GET_PARAMETER (PARAMETER_NAME, PVT, VALUE_LIST, STATUS)
{
{ PARAMETER_NAME:  (input)  This  parameter specifies any one of the parameter
{       names for the parameter in question.
{
{ PVT: (input) This parameter specifies the Parameter Value Table for the
{       parameter list.
{
{ VALUE_LIST: (output) This parameter specifies the parameter's value list.
{
{ STATUS: (output) This parameter specifies the request status.

  PROCEDURE [XDCL, #GATE] clp_get_parameter ALIAS 'clpgpa' ( {
       parameter_name: string ( * );
       pvt: clt$parameter_value_table;
    VAR value_list: ost$string;
    VAR status: clt$status);
```

CLP GET PARAM LIST

```
{ Procedure Name:  clp_get_param_list
{
{   The purpose of this request is to return the entire parameter list, in its
{ uninterpreted form, as a string.  If no parameters were given, a null string
{ is returned.
{
{ Call Format:
{       (*callc clxgpl)
{       CLP_GET_PARAM_LIST (PARAMETER_LIST, PVT, STATUS)
{
{ PARAMETER_LIST: (output) This parameter specifies the parameter list.
{
{ PVT: (input) This parameter specifies the Parameter Value Table for the
{       parameter list.
{
{ STATUS: (output) This parameter specifies the request status.

  PROCEDURE [XDCL, #GATE] clp_get_param_list ALIAS 'clpgpl' ( {
    VAR parameter_list: ost$string;
        pvt: clt$parameter_value_table;
    VAR status: clt$status);
```

CLP GET SET COUNT

```
{ Procedure Name:  clp_get_set_count
{
{   The purpose  of  this  request  is  to  determine the number of value sets
{ supplied for a particular parameter in the actual parameter  list.   If  the
{ parameter in  question was not given, a value set count of zero is returned.
{{ Call Format:
{       (*callc clxgsc)
{       CLP_GET_SET_COUNT (PARAMETER_NAME, PVT, VALUE_SET_COUNT, STATUS)
{
{ PARAMETER_NAME:  (input)  This  parameter specifies any one of the parameter
{       names for the parameter in question.
{
{ PVT: (input) This parameter specifies the Parameter Value Table for the
{       parameter list.
{
{ VALUE_SET_COUNT: (output) This parameter specifies the number of value  sets
{       given for the parameter.
{
{ STATUS: (output) This parameter specifies the request status.

  PROCEDURE [XDCL, #GATE] clp_get_set_count ALIAS 'clpgsc' ( {
        parameter_name: string ( * );
        pvt: clt$parameter_value_table;
    VAR value_set_count: 0 .. clc$max_value_sets;
    VAR status: clt$status);
```

CLP GET VALUE

```
{ Procedure Name:  clp_get_value
{
{   The purpose  of this request is to get a parameter value that was given in
{ the actual parameter list.  If the requested value was not given, a value of
{ kind  "unknown"  is  returned.   If the request is for the "high" value of a
{ range and a high value was not supplied but a "low" value was, then the  low
{ value is returned.
{
{ Call Format:
{       (*callc clxgval)
{       CLP_GET_VALUE (PARAMETER_NAME, PVT, VALUE_SET_NUMBER, VALUE_NUMBER,
{         LOW_OR_HIGH, VALUE, STATUS)
{
{ PARAMETER_NAME: (input) This parameter specifies any one  of  the  parameter
{       names for the parameter in question.
{
{ PVT: (input) This parameter specifies the Parameter Value Table for the
{       parameter list.
{
{ VALUE_SET_NUMBER:  (input) This parameter specifies from which value set the
{       value is to be obtained.
{
{ VALUE_NUMBER: (input) This parameter specifies which value within the  value
{       set is to be obtained.
{
{ LOW_OR_HIGH:  (input) This parameter specifies which "side" of a value range
{       is to be obtained.
{
{ VALUE: (output) This parameter specifies the parameter value.
{
{ STATUS: (output) This parameter specifies the request status.

  PROCEDURE [XDCL, #GATE] clp_get_value ALIAS 'clpgva' ( {
        parameter_name: string ( * );
        pvt: clt$parameter_value_table;
        value_set_number: 1 .. clc$max_value_sets;
        value_number: 1 .. clc$max_values_per_set;
        low_or_high: clt$low_or_high;
    VAR value: clt$value;
    VAR status: clt$status);
```

CLP GET VALUE COUNT

```
{ Procedure Name:  clp_get_value_count
{
{   The purpose  of this request is to determine the number of values given in
{ a particular value set for a particular parameter in  the  actual  parameter
{ list.   If  the  requested value set was not given, a value count of zero is
{ returned.
{
{ Call Format:
{       (*callc clxgvc)
{       CLP_GET_VALUE_COUNT (PARAMETER_NAME, PVT, VALUE_SET_NUMBER,
{         VALUE_COUNT, STATUS)
{
{ PARAMETER_NAME:  (input)  This  parameter specifies any one of the parameter
{       names for the parameter in question.
{
{ PVT: (input) This parameter specifies the Parameter Value Table for the
{       parameter list.
{
{ VALUE_SET_NUMBER:  (input)  This  parameter  specifies  the  value  set   in
{       question.
{
{ VALUE_COUNT: (output) This parameter specifies the number of values given in
{       the specified value set for the specified parameter.
{
{ STATUS: (output) This parameter specifies the request status.

  PROCEDURE [XDCL, #GATE] clp_get_value_count ALIAS 'clpgvc' ( {
        parameter_name: string ( * );
        pvt: clt$parameter_value_table;
        value_set_number: 1 .. clc$max_value_sets;
    VAR value_count: 0 .. clc$max_values_per_set;
    VAR status: clt$status);
```

CLP PARSE COMMAND

```
{ Procedure Name:  clp_parse_command
{
{   The purpose of this request is to parse a command into its major component
{ parts.
{
{ Call Format:
{       (*callc clxpcom)
{       CLP_PARSE_COMMAND (COMMAND, NAME_INDEX, NAME_SIZE,
{         NAME, SEPARATOR, PARAMETER_LIST, EMPTY_COMMAND, STATUS)
{
{ COMMAND: (input) This parameter specifies the command to be parsed.
{
{ NAME_INDEX: (output) This parameter specifies the position within COMMAND
{       where the command reference begins. It is the beginning of the
{       command's name.  (Undefined if empty_command is true.)
{
{ NAME_SIZE:  (output)  This  parameter  specifies  the  size  of  (number  of
{       characters in) the command reference.  It is the size of the command's
{       name.  (Undefined if empty_command is true.)
{
{ NAME: (output) This parameter specifies the name of the  command returned in
{       upper case.  (Undefined if empty_command is true.)
{
{ SEPARATOR:  (output)  This  parameter  specifies  the  separator between the
{       command  reference, and  the  parameters  for  the  command.
{       Possible  values  are:  clc$space_token,  clc$comma_token  and
{       clc$eol_token.
{       (Undefined if empty_command is true.)
{
{ PARAMETER_LIST: (output) This parameter specifies the  command's  parameters
{       in the form of a string.
{       (Undefined if empty_command is true.)
{
{ EMPTY_COMMAND: (output) This parameter  specifies  whether  the  command  is
{       empty (consists solely of spaces and/or comments).
{
{ STATUS: (output) This parameter specifies the request status.

  PROCEDURE [XDCL, #GATE] clp_parse_command ALIAS 'clppcom' ( {
        command: string ( * );
    VAR name_index: ost$string_index;
    VAR name_size: ost$string_size;
    VAR name: clt$name;
    VAR separator: clt$lexical_kinds;
    VAR parameter_list: ost$string;
    VAR empty_command: boolean;
    VAR status: clt$status);
```

CLP PARSE TERMINATE

```
{ Procedure Name:  clp_parse_terminate
{
{   The purpose of this request is to free the PVT related memory that was
{ allocated during the parsing of a command.
{
{ Call Format:
{       (*callc clxptrm)
{       CLP_PARSE_TERMINATE (PVT, STATUS)
{
{ PVT:  (input) The PVT contains the pointers to all of the memory areas
{       which were allocated during the parsing of this command.
{
{ STATUS: (output) This parameter specifies the request status.

  PROCEDURE [XDCL, #GATE] clp_parse_terminate ALIAS 'clpptrm' ( {
    VAR pvt: clt$parameter_value_table;
    VAR status: clt$status);
```

CLP PROCESS COMMAND

```
{ Procedure Name:  clp_process_command
{
{ Purpose:  Issue command string to Command M-E
{
{ Description:
{  This common function accepts a character string and converts it to
{  management data unit syntax. It is then sent to the Command M-E via
{  intertask message.  We wait until it returns our command response,
{  also via intertask message.
{
{  A common use of this routine is for issuing internally generated
{  commands.
{
{ Call Format:
{       (*callc clxpcm)
{       clp_process_command (str, c_code, response);
{
{ Entry Conditions
{  str := command string to be processed
{
{ Exit Conditions
{  response:  pointer to buffer containing command
{             response data unit syntax
{  c_code:  condition code
{
{ Limitations
{  Any intertask messages the caller expects to receive will be
{  discarded if received by clp_process_command.

  PROCEDURE [XDCL, #GATE] clp_process_command ( {
       str: ost$string;
    VAR status: clt$status);
```

CLP SCAN PARAMETER LIST

```
{ Procedure Name:  clp_scan_parameter_list
{
{   The purpose  of  this  request is to scan the parameter list for a command
{ under control of a Parameter Descriptor Table.  This  request  may  only  be
{ invoked once an environment for the parameter list has been established.  An
{ environment is established automatically for a command processor,  but  must
{ be  explicitly  created (via clp_push_parameters) for a program other than a
{ command processor, or for a command  processor  which  wants  to  have  some
{ string interpreted as a parameter list.
{
{ Call Format:
{       (*callc clxspl)
{       CLP_SCAN_PARAMETER_LIST (PARAMETER_LIST, PDT, PVT, STATUS)
{
{ PARAMETER_LIST:  (input)  This  parameter specifies the parameter list to be
{       scanned.   Normally,   this   is   the   paramter   passed   to   a   command
{       processor.     The    contents    of    this    sequence    is    described    by
{       ost$string.
{
{ PDT: (input) This parameter specifies the Parameter Descriptor Table for the
{       parameter list.
{
{ PVT: (output) This parameter specifies the Parameter Variable table for the
{       parameter list.
{
{ STATUS: (output) This parameter specifies the request status.

  PROCEDURE [XDCL, #GATE] clp_scan_parameter_list ALIAS 'clpscpl' ( {
        parameter_list: ost$string;
        pdt: clt$parameter_descriptor_table;
    VAR pvt: clt$parameter_value_table;
    VAR status: clt$status);
```

CLP TEST PARAMETER

```
{ Procedure Name:  clp_test_parameter
{
{ The purpose  of this request is to test whether a particular parameter was
{ specified in the actual parameter list.
{
{ Call Format:
{       (*callc clxtpar)
{       CLP_TEST_PARAMETER (PARAMETER_NAME, PVT, PARAMETER_SPECIFIED, STATUS)
{
{ PARAMETER_NAME: (input) This parameter specifies any one  of  the  parameter
{       names for the parameter in question.
{
{ PVT: (input) This parameter specifies the Parameter Value Table for the
{       parameter list.
{
{ PARAMETER_SPECIFIED:  (output)  This  parameter  specifies the result of the
{       test:
{       TRUE - the parameter was given,
{       FALSE - the parameter was not given.
{
{ STATUS: (output) This parameter specifies the request status.

  PROCEDURE [XDCL, #GATE] clp_test_parameter ALIAS 'clptsp' ( {
        parameter_name: string ( * );
        pvt: clt$parameter_value_table;
    VAR parameter_specified: boolean;
    VAR status: clt$status);
```

CLP TEST RANGE

```
{ Procedure Name:  clp_test_range
{
{   The purpose of this request is to determine whether a particular value for
{ a particular parameter was given as a range.  If the requested value was not
{ given, then false is returned.
{
{ Call Format:
{       (*callc clxtrng)
{       CLP_TEST_RANGE (PARAMETER_NAME, PVT, VALUE_SET_NUMBER, VALUE_NUMBER,
{         RANGE_SPECIFIED, STATUS)
{
{ PARAMETER_NAME: (input) This parameter specifies any one  of  the  parameter
{       names for the parameter in question.
{
{ PVT: (input) This parameter specifies the Parameter Value Table for the
{       parameter list.
{
{ VALUE_SET_NUMBER:   (input)  This  parameter  specifies  the  value  set  in
{       question.
{
{ VALUE_NUMBER: (input) This parameter specifies the value in question.
{
{ RANGE_SPECIFIED: (output) This parameter specifies the result of the test.
{
{ STATUS: (output) This parameter specifies the request status.

  PROCEDURE [XDCL, #GATE] clp_test_range ALIAS 'clptsr' ( {
        parameter_name: string ( * );
        pvt: clt$parameter_value_table;
        value_set_number: 1 .. clc$max_value_sets;
        value_number: 1 .. clc$max_values_per_set;
    VAR range_specified: boolean;
    VAR status: clt$status);
```

CLP TRIMMED STRING SIZE

```
{ Function Name:  clp_trimmed_string_size
{
{ Description:
{   The purpose  of  this  function  is  to  return  the size of a string once
{ trailing space characters have been removed from  it.   The  horizontal  tab
{ (HT)  in addition to the space are considered to be space characters by this
{ function.
{
{       (*callc clxtss)
{       CLP_TRIMMED_STRING_SIZE (STR): TRIMMED_STRING_SIZE
{
{ STR: (input) This parameter specifies the string for which the trimmed  size
{       is to be returned.

  FUNCTION [XDCL, #GATE] clp_trimmed_string_size ALIAS 'clptss' ( {
    str: string ( * )) {
       : ost$string_size;
```

CONVERT INTEGER TO POINTER

```
{ PROCEDURE NAME:  convert_integer_to_pointer
{
{ PURPOSE:
{   convert integer to pointer.
{
{ CALL FORMAT:
{   (*callc CMIPCIP)
{   address := convert_integer_to_pointer (number);
{
{ DESCRIPTION:
{   Provides a needed function for users who need to do pointer arithmetic.
{   (Probably should be restricter to hardware interface routines.)
{

  FUNCTION [XDCL] convert_integer_to_pointer (val: integer): ↑cell;
```

CONVERT POINTER TO INTEGER

```
{ PROCEDURE NAME:  convert_pointer_to_integer
{
{ PURPOSE:
{   Convert pointer to integer.
{
{ CALL FORMAT:
{   (*callc CMIPCPI)
{   number := convert_pointer_to_integer (address);
{
{ DESCRIPTION:
{   Provides a needed function for users who need to do pointer arithmetic.
{   (Probably should be restricter to hardware interface routines.)
{

  FUNCTION [xdcl] convert_pointer_to_integer (val: ↑cell): integer;
```

COPY

```
{ PROCEDURE NAME: copy
{
{ PURPOSE:
{   Logical copy of Message To New Buffer Chain.
{
{ CALL FORMAT:
{   (*callc CMXPCPY)
{   copy (message, threshold);
{
{ DESCRIPTION:
{   The message is logically copied to new buffers, and
{   the old set of buffers is released.
{
{
{ NOTES:
{   "message" must be a valid buffer chain address
{
{

  PROCEDURE [XDCL] copy ( {
      VAR message: buf_ptr; {the message to be copied
          threshold: threshold_size); { threshold for buffer acquisition
```

CLEAR ALLOCATE CONDITIONAL

```
{ PROCEDURE NAME:   CLEAR_ALLOCATE_CONDITIONAL
{
{ PURPOSE:
{   Conditionally allocate memory from the system heap, and clear it (to zeros)
{
{ DESCRIPTION:
{   This procedures attempts allocation of memory, if the memory is
{   obtained, it is cleared and the pointer to the allocated memory is
{   returned. Otherwise NIL will be returned if the allocation failed.
{
{ NOTES:
{   The allocated memory will always be an even number of bytes,
{   and start at an even byte boundary.
{

  PROCEDURE [INLINE] clear_allocate_conditional ( {
        memory_bytes: 1 .. 32766) {:} ^cell;
```

DATA REQUEST 3A

```
{  PROCEDURE NAME:  data_request_3a
{
{  PURPOSE:
{    This procedure is provided by Intranet to allow users to send a datagram
{    downline via a direct call.
{
{  DESCRIPTION:
{    A user of Intranet calls the data_request_3a procedure directly.  The
{    user must provide the network_id the datagram is to be transmitted on,
{    the address of the destination system, its associated SAP identifier
{    (returned on the open_3a_sap request), and the address of the datagram.
{    Intranet determines that a SAP was opened by the user for the network_id
{    specified and obtains the address of the associated NIB.  Intranet then
{    builds the 3A header and places the associated header information
{    for the type of network solution specified, enqueues the datagram in
{    the associated 3A queue, changes the network status if the network
{    solution becomes congested, notifies all users of 3A of any status
{    changes and sends an intertask message to the SSR if it is not currently
{    active.
{
{  RETURNS:
{    Name               Type     Description
{    data_ptr           buf_ptr  This parameter contains the user datagram to be
{                                transmitted downline.
{                                If the request was successful, then it's
{                                returned as NIL to the Intranet user to ensure
{                                that the data_ptr queued is not inadvertently
{                                modified by the user.
{                                If the request was unsuccessful, then the
{                                datagram originally passed to 3a is returned
{                                to the user.
{
{    request_processed boolean  The status of the data request is returned to
{                                the user via this parameter.
{
{  GLOBAL DATA REFERENCED:
{    network_solution_list
{    sap_table
{
{  GLOBAL DATA MODIFIED:
{    sap_table
{
{  NOTES AND CAUTIONS:
{    An intertask message is sent to the SSR associated with the LIB if the
{    SSR is not currently retrieving datagrams from the 3A queue.

  PROCEDURE data_request_3a ( {
        network_id: network_id_type;
        destination_address: system_id_type;
        sap: intranet_sap_type;
    VAR data_ptr: buf_ptr;
    VAR request_processed: boolean);
```

DATA 3A REQUEST

```
{ PROCEDURE NAME:  data_3a_request
{
{ PURPOSE:
{   Sends datagrams to other 3B users.
{
{ DESCRIPTION
{   The source and destination SAPs are verified not to be defaults.
{   The source SAP is checked to verify that it is open.  bld_3bhdr
{   is called to build the 3B header from the supplied parameters
{   and to prefix it to the data.  If bld_3bhdr is successful, routing
{   is called to determine how the 3b_pdu is to reach its destination.
{
{ GLOBAL INPUT:
{   none
{
{ GLOBAL OUTPUT:
{   none
{

  PROCEDURE [XDCL] data_3b_request ( {
        req_param: ↑internet_req_if; { INPUT - request parameters
    VAR return_code: internet_return_codes); { OUTPUT
```

DEAD STOP

```
{ PROCEDURE NAME:  dead_stop
{
{ PURPOSE:  Dead Stop.
{
{ CALL FORMAT:
{   (*callc cmxpded)
{   dead_stop (halt_code);
{
{ DESCRIPTION:
{   This procedure calls di_reset with a reset code of software_dead_stop.
{

  PROCEDURE [XDCL] dead_stop (halt_code: integer);
```

DECREMENT MODULE USE COUNT

```
{ PROCEDURE NAME:  decrement_module_use_count
{
{ PURPOSE:
{   decrement the module use count
{
{ CALL FORMAT:
{   *callc dlxdmuc
{   decrement_module_use_count(entry_point_name, entry_point_found);
{
{ DESCRIPTION:
{   The module use count of the indicated entry point is decremented.
{ If the count becomes zero, then the module is made available for deload.
{ A task abort is caused if the counter becomes negative.  If the given
{ entry point name is all blanks, then the module use count of the first
{ module of the running task is decremented.  This procedure is used when
{ the module use count was previously incremented and procedure stop_task
{ will not be called to decrement the counter.
{

  PROCEDURE [XDCL] decrement_module_use_count
    ({
         entry_point_name: pmt$program_name;
     VAR entry_point_found: boolean);
```

DELAY PROCESSING

```
{ PROCEDURE NAME: delay_processing
{
{ PURPOSE:
{   delay a task for a period of time
{
{ CALL FORMAT:
{   (*callc CMXPDLY)
{   delay_processing(hours,minutes,seconds,milliseconds);
{
{ PURPOSE:  delay Processing for a Period of Time.  This
{   routine may be called whenever someone wants to delay
{   processing for a finite period, such as a timeout mechanism.
{   A normal return occurs when processing is resumed.
{
{ NOTES AND CAUTIONS:
{   Note that the Executive Call After Interval service is used
{   to restart the task.  The Executive guarantees that the
{   requestor will wait at least as long as requested, but does
{   not guarantee a maximum period.  Thus,
{   delay_processing(0,0,0,200) will delay at least 200
{   milliseconds, but may delay longer, even up to several
{   seconds in a very busy system.  Note also that the Executive
{   Wait/Wakeup service is utilized by this routine.  Thus, any
{   event that presents a Wakeup to the task will produce a
{   Wakeup from this routine.  The routine will cancel the
{   outstanding timing request if that occurs, allowing this
{   routine to be used as "delay processing until <timeout> OR
{   <wake-up event>".
{

  PROCEDURE [XDCL] delay_processing ( {
        hours: 0 .. 24;
        minutes: 0 .. 59;
        seconds: 0 .. 59;
        milliseconds: 0 .. 999);
```

DIR ABORT

```
{     PROCEDURE NAME:  dir_abort
{
{     PURPOSE:
{       Abort an outstanding Translation Request.  This request
{       was issued with the procedure dir_translate.
{
{     DESCRIPTION:
{       This procedure scans the Translation Request Data Store
{       (TRDS)  to  locate the outstanding translation request.
{       The entry is deleted if it is not active.
{
{     CALL FORMAT:
{       (*callc drxdir)
{       DIR_ABORT (TRANSLATION_REQUEST_IDENTIFIER, STATUS);
{
{       TRANSLATION_REQUEST_IDENTIFIER:       (input)      This
{       parameter was returned by the Directory at the time  of
{       the  translation request.  It must be supplied to abort
{       this request.
{
{       STATUS: (output)This parameter is returned. Values are:
{       dir_abort_ok - Successful abort of translation
{       dir_abort_err - No Translation request found.
{
```

DIR CHANGE

```
{     PROCEDURE NAME:  dir_change
{
{     PURPOSE:
{       Change attributes for an existing Directory Entry.
{
{     DESCRIPTION:
{       The  caller must supply the title, password, and
{       directory entry identifier.  The entry is updated in
{       the Registered Data Store based on  fields specified in
{       the change_effectors_set.  The priority and user
{       information may be changed.
{
{     CALL FORMAT:
{       (*callc drxdir)
{       DIR_CHANGE (REGISTRATION_CONTROL_BLOCK,
{              CHANGE_EFFECTORS_SET, DIRECTORY_ENTRY_IDENTIFIER,
{              STATUS);
{
{       REGISTRATION_CONTROL_BLOCK:    (input)  This  record
{       specifies  parameters needed  to  change  the title:
{       .title_ptr - Pointer to the title
{       .community_ptr - Not used.
{       .password - Password. Must be supplied to change
{           the title.
{       .address - Not used.
{       .userinfo_ptr - Pointer to optional user information
{       .priority - Priority of the title (1..0ff(16))
{       .service - Not used.
{       .translation_domain - Not used
{       .distribute_title - Not used.
{       .class - Not used.
{
{       CHANGE_EFFECTORS_SET:   (input)  Elements  in  this set
{       must  be  set  in  order  to  change  the corresponding
{       attributes.
{
{       DIRECTORY_ENTRY_IDENTIFIER:    (input)  This  parameter
{       was returned when the title was  registered.  It  must
{       be supplied to change the title.
{
{       STATUS: (output)This parameter is returned. Values are:
{         dir_change_ok      successful change
{         dir_no_room        Allocate failed
{         dir_title_err      title length > max_title_len or = 0
{         dir_userinfo_err   userinfo > max_userinfo_len
{         dir_entry_not_found   No entry with matching title,
{                            password, and Directory Entry ID.
{
{     ENTRY CONDITIONS:
{       The  REGISTRATION_CONTROL_BLOCK  must  be initially set
{       to defaults via the inline procedure DIR_RCB_INIT.
{
```

DIR CREATE

```
{     PROCEDURE NAME:  dir_create
{
{     PURPOSE:
{       Register a title in the Directory.
{
{     DESCRIPTION:
{       This  procedure  creates  a directory entry for a given
{       title and address.  The entry  is put in the Registered
{       Data Store (RDS).
{
{     CALL FORMAT:
{       (*callc drxdir)
{       DIR_CREATE (REGISTRATION_CONTROL_BLOCK,
{             DIRECTORY_ENTRY_IDENTIFIER, STATUS);
{
{       REGISTRATION_CONTROL_BLOCK:    (input)  This  record
{       specifies  all the parameters needed  to  register  the
{       title:
{       .title_ptr - Pointer to the title
{       .community_ptr - Pointer to array of communities if
{           the Translation Domain specifies list_of_communities
{       .password - Password. Must be supplied to change
{           or delete the title.
{       .address - Address associated with the title
{       .userinfo_ptr - Pointer to optional user information
{       .priority - Priority of the title (1..0ff(16))
{       .service - Next layer software used by this title.
{       .translation_domain - Domain where title may be
{           translated.
{       .distribute_title - Boolean set to distribute the
{           title over the translation domain.
{       .class - Internal or External CDNA title.
{
{       DIRECTORY_ENTRY_IDENTIFIER:   (output)  This  parameter
{       is  returned  by the Directory.  It uniquely identifies
{       this registered  title throughout the catenet.  It must
{       be supplied to change or delete this title.
{
{       STATUS: (output)This parameter is returned. Values are:
{         dir_create_ok     successful registration
{         dir_no_room       Allocate failed
{         dir_duplicate     Title & Address already registered
{         dir_title_err     title length > max_title_len or = 0
{         dir_address_err   address type is incorrect
{         dir_userinfo_err  userinfo > max_userinfo_len
{         dir_community_err communities > max_community_titles
{
{     ENTRY CONDITIONS:
{       The  REGISTRATION_CONTROL_BLOCK  must  be initially set
{       to defaults via the inline procedure DIR_RCB_INIT.
{
```

DIR DELETE

```
{     PROCEDURE NAME:  dir_delete
{
{     PURPOSE:
{       Delete  an  existing  Directory Entry.
{
{     DESCRIPTION:
{       This  procedure  deletes  an  existing  directory entry
{       given   a   title,   password,   and   directory  entry
{       identifier.   The entry is deleted from the  Registered
{       Data Store.
{
{     CALL FORMAT:
{       (*callc drxdir)
{       DIR_DELETE (TITLE_PTR, PASSWORD,
{             DIRECTORY_ENTRY_IDENTIFIER, STATUS);
{
{       TITLE_PTR: (input) This parameter points to the title.
{
{       PASSWORD: (input) This password was set at registration.
{           It must be supplied to delete the title.  Note, the
{           default password is an integer zero (0).
{
{       DIRECTORY_ENTRY_IDENTIFIER:    (input)  This  parameter
{       was returned when the title was  registered.   It  must
{       be supplied to delete the title.
{
{       STATUS: (output)This parameter is returned. Values are:
{         dir_delete_ok      successful delete
{         dir_no_room        Allocate failed
{         dir_title_err      title length > max_title_len or = 0
{         dir_entry_not_found   No entry with matching title,
{                            password, and Directory Entry ID.
{
```

DIR_PURGE

```
{     PROCEDURE NAME:  dir_purge
{
{     PURPOSE:
{       Purge  a  Directory  Entry  from  the  Translation Data
{       Store.
{
{     DESCRIPTION:
{       This procedure locates the Translation Data Entry with
{       same title and Directory Entry Identifier.  It deletes
{       this entry.  Note, the user calls this procedure after
{       a connection attempt fails and the user does not want
{       another indication with this entry.
{
{     CALL FORMAT:
{       (*callc drxdir)
{       DIR_PURGE (TITLE_PTR, DIRECTORY_ENTRY_IDENTIFIER,
{              STATUS);
{
{       TITLE_PTR: (input) This parameter points to the title.
{
{       DIRECTORY_ENTRY_IDENTIFIER:   (input)  This  parameter
{       was returned  by the Directory when the title was
{       translated.  It must be supplied to purge this title.
{
{       STATUS: (output)This parameter is returned. Values are:
{         dir_purge_ok      successful purge of the title
{         dir_entry_not_found Title not in Directory cache.
{
```

DIR TRANSLATE

```
{     PROCEDURE NAME:  dir_translate
{
{     PURPOSE:
{       Return  one  or  more  title translations for the given
{       title.  Resume control immediately.
{
{     DESCRIPTION:
{       This  routine  is called by the users to request one or
{       more  title  translations.   The  user  is  resumed
{       immediately  with  the success/fail of the request
{       returned  in dir_status.   Each  individual  title
{       translation  indication  is  returned  to  the user's
{       ↑PROCEDURE.  The  user may abort this Translation
{       Request with a parameter on the ↑PROCEDURE.  The
{       search for titles may be active or passive.  If
{       active, the Translation Request can  be terminated by
{       the  user  or  by  time  expiration.  If passive, the
{       Translation Request is only terminated by the user.
{
{     CALL FORMAT:
{       (*callc drxdir)
{       DIR_TRANSLATE (TRANSLATION_CONTROL_BLOCK,
{              TRANSLATION_REQUEST_IDENTIFIER, STATUS);
{
{       TRANSLATION_CONTROL_BLOCK:    (input)  This  record
{       specifies  all  the  parameters  needed  to translate a
{       title:
{       .title_ptr - Pointer to the title
{       .community_ptr - Pointer to array of communities if
{             the Search Domain specifies list_of_communities
{       .user_id - Supplied by user.  Returned to the user's
{             translation_if.
{       .translation_if - ↑Procedure where indications are
{             returned. Parameters are  the
{             TRANSLATION_INDICATION_CONTROL_BLOCK and
{             and ABORT_TRANSLATION_REQUET.
{       .time - Time duration of search in seconds.
{             Not used for passive search.
{       .service - Service must match registered title's
{             service if not dir_unknown.
{       .search_domain - Domain where the title may be
{             registered.
{       .recurrent_search - If FALSE, translation is
{             terminated by the user or by time expired.
{             If TRUE, user must terminate search.
{       .class - Class must match registered title's class.
{       .wild_card - Title may contain wild card characters.
{
{       TRANSLATION_REQUEST_IDENTIFIER: (output) This parameter
{       is  returned  by the Directory.  It uniquely identifies
{       this translation  request  in  this  system.  It must be
{       supplied to wait for translation  termination  or abort
```

```
{       the request.
{
{       STATUS: (output)This parameter is returned. Values are:
{       dir_translate_ok - successful translation call. Note,
{             indications are returned to translation_if.
{       dir_no_room - Allocate failed
{       dir_title_err - Title length > max_title_len or = 0
{       dir_community_err - Communities > max_community_titles
{       dir_translation_if_err - ↑Procedure was not supplied.
{
{     ENTRY CONDITIONS:
{       The  TRANSLATION_CONTROL_BLOCK must be initially set to
{       defaults via the inline procedure DIR_TCB_INIT.
{
```

DIR TRANSLATE AND WAIT

```
{     PROCEDURE NAME:  dir_translate_and_wait
{
{     PURPOSE:
{       Return one translation for the given title.  Wait until
{       the translation has completed or the time expires.
{
{     DESCRIPTION:
{       This routine is called by the users to request only one
{       title translation.  This request causes an active
{       search of the search domain to locate the title.  The
{       first title located is returned.  The user is suspended
{       until the call has been processed.  The confirm/reject
{       is returned in dir_status at the RETURN.  If dir_status
{       = dir_title_found, then the Directory Entry Id,
{       address, userinfo, priority, and service are returned
{       in the dir_ttcb record.  If dir_status =
{       dir_time_expired, the title was not found before the
{       user time limit expired.  Other dir_status codes
{       indicate a user error in call setup or no room to
{       create the Translation Request Data Store entry.
{
{     CALL FORMAT:
{       (*callc drxdir)
{       DIR_TRANSLATE_AND_WAIT (TRANSLATION_CONTROL_BLOCK,
{             TITLE_TRANSLATION_CONTROL_BLOCK, STATUS);
{
{       TRANSLATION_CONTROL_BLOCK:    (input) This record
{       specifies all the parameters needed to translate a
{       title:
{       .title_ptr - Pointer to the title
{       .community_ptr - Pointer to array of communities if
{           the Search Domain specifies list_of_communities
{       .user_id - Not used.
{       .translation_if - Set to NIL.
{       .time - Time duration of search in seconds.
{       .service - Service must match registered title's
{           service if not dir_unknown.
{       .search_domain - Domain where the title may be
{           registered.
{       .recurrent_search - Set FALSE.
{       .class - Class must match registered title's class.
{       .wild_card - Title may contain wild card characters.
{
{       TITLE_TRANSLATION_CONTROL_BLOCK: (output) This record
{       is returned by the Directory if status =
{       dir_title_found.  Values are:
{       .dir_id - Directory Entry Identifier for this title.
{       .address - Address registered for this title.
{       .userinfo - User information registered for this title.
{       .priority - Current priority for this title
{       .service - Next layer software used by this title.
{
```

```
{       STATUS: (output)This parameter is returned. Values are:
{       dir_title_found - Title translation returned.
{       dir_time_expired - No Title translation before time limit.
{       dir_no_room - Allocate failed
{       dir_title_err - Title length > max_title_len or = 0
{       dir_community_err - Communities > max_community_titles
{
{     ENTRY CONDITIONS:
{       The  TRANSLATION_CONTROL_BLOCK must be initially set to
{       defaults via the inline procedure DIR_TCB_INIT.
{
{     CAUTION:
{       Dir_Translate_and_Wait     uses    the    EXEC    calls
{       WAIT/WAKE_UP.  If the caller  uses WAIT/WAKE_UP, a flag
{       bit must be set and checked to assure  the  caller  was
{       woke up.
{
```

DIR WAIT

```
{     PROCEDURE NAME:  dir_wait
{
{     PURPOSE:
{       Give  up  control  of  the  CPU  until  the  Directory
{       Translation Request has terminated.
{
{     DESCRIPTION:
{       This  procedure  verifies  there  is  an  outstanding
{       Translation  Request  with  this  Translation  Request
{       Identifier.  The running  task  is  put  to sleep.  The
{       task is resumed when the translation request  has  been
{       terminated.
{
{     CALL FORMAT:
{       (*callc drxdir)
{       DIR_WAIT (TRANSLATION_REQUEST_IDENTIFIER);
{
{       TRANSLATION_REQUEST_IDENTIFIER:       (input)      This
{       parameter was returned by the Directory at the time  of
{       the  translation  request.  It must be supplied to wait
{       for translation termination.
{
{     CAUTION:
{       Dir_Wait  uses  the  EXEC  calls  WAIT/WAKE_UP.  If the
{       caller uses WAIT/WAKE_UP, a flag bit  must  be  set and
{       checked to assure the caller was woke up.
{
```

DI_DEBUG

```
{ PROCEDURE NAME: di_debug
{
{ PURPOSE: Stop in the DI_DEBUGGER from user program.
{
{ CALL FORMAT:
{   *callc CMXDBUG
{    di_debug;
{
{ DESCRIPTION:
{   If the DI debug program has not been initialized or it was initialized
{   through a di_debug_init request the DI debug program will be set to stop on
{   all DI debug detected errors. The DI debug program will then enter its main
{   console input loop waiting for debug commands to be entered through the
{   DI console.
{
{   If the DI debug program has already been initialized by a di_debug request
{   it will retain its current trap state and enter the main console input
{   loop waiting for debug commands to be entered through the DI console.
{
 PROCEDURE [XREF] di_debug;
```

DI DEBUG INIT

```
{ PROCEDURE NAME: di_debug_init
{
{ PURPOSE: Initialize the Di debug program to console break mode only.
{
{ CALL FORMAT:
{   *callc CMXDBUG
{    di_debug_init;
{
{ DESCRIPTION:
{   The DI debug program will be initialized to trap the DI console break
{   input only. If the DI debug program has already been initialized this
{   request will be ignored leaving the DI debug program in its current
{   trap state.
{
{   If the DI debug program is initialized in this state it will enable
{   all of the available debug program error stops on the first occurrance
{   of a trap resulting from the DI console break.
{
 PROCEDURE [XREF] di_debug_init;
```

DUMP_CLOSE

```
{ PROCEDURE NAME: dump_close
{
{ PURPOSE: indicate done supplying dump information
{
{ CALL FORMAT:
{  (*callc(cmxsisa)
{  dump_close(dump_identifier);
{
{ DESCRIPTION:
{ The dump task is sent a message indicating that the user is done
{ supplying dump information.
{
{ NOTE - if the dump identifier is not valid, then no message will be
{        sent and the caller returned to.
{
{ SEE ALSO:
{ set_recovery_procedure
{ reset_recovery_procedure
{ dump_write
{

  PROCEDURE [XDCL, #GATE] dump_close (
    sa_dump_identifier: ↑cell); { address of dump control block
```

DUMP WRITE

```
{ PROCEDURE NAME: dump_write
{
{ PURPOSE: move user dump information to dump buffer chain
{
{ CALL FORMAT:
{  (*callc(cmxsisa)
{  dump_write(dump_identifier,dump_address,dump_byte_count,threshold);
{
{ DESCRIPTION:
{ A header and dump information is appended to the dump buffer chain
{ associated with the dump identifier.  If the total number of bytes in the
{ buffer chain is above the maximum allowed in a dump buffer chain, then
{ message(s) will be sent to the dump task identifying buffers to be
{ immediately written to the dump file.
{
{ NOTE - If the dump identifier is not valid, then the dump information
{        will be discarded and the caller returned to.
{
{ SEE ALSO:
{ set_recovery_procedure
{ reset_recovery_procedure
{ dump_close
{

  PROCEDURE [XDCL, #GATE] dump_write (
          sa_dump_identifier: ↑cell, { address of  dump control block
          dump_address: ↑cell, { address of information to dump
          dump_byte_count: sat$max_dump_size, { number of bytes to dump
          threshold: threshold_size); { threshold with which to obtain buffers
```

EXECUTIVE ERROR TABLE

```
{ TABLE NAME: Executive error table
{
{ PURPOSE:
{   Describes Executive error table.  This table is initialized by the
{   system executive and is located in mpb ram.
{
{       ************************************************************
{       *                     -----NOTICE-----                     *
{       *   exec_error_table is interdependent with deck  "EXDERTB".  *
{       *   Any changed to "CMCERTB' or "EXDERTB" should result in    *
{       *   corresponding modifications to the other deck.           *
{       *                     -----NOTICE-----                     *
{       ************************************************************
{
{
{
{ CALL FORMAT:
{   (*callc CMCERTB
{

   TYPE
     executive_error_table = record
       stop_supervisor_stack_pointer: ↑supervisor_pc_rec,
       last_error_address: ↑error_buffer,
       lock_last_error: 0 .. 0ffff(16), {last_error_address being updated
       address_error_being_processed: 0 .. 0ffff(16),
       number_of_spurious_interrupts: 0 .. 0ffff(16),
       smm_error_count: array[ 0 .. 7 ] of 0 .. 0ffff(16),
       total_error_count: 0 .. 0ffff(16),
       system_ancestor_tcb: task_ptr,
       debug_address_called_on_error: ↑cell,
       error_buffers: array[ 0 .. number_of_error_buffers ] of error_buffer,
     recend;
```

```
TYPE
  error_buffer = record
    executive_error_code: ex_error_codes,
    lock_error_buffer: 0 .. 0ffff(16),{non-zero to lock error buffer
    binclock_at_time_of_error: integer,
    d0_thru_d7: array[ 0 .. 7 ] of integer,
    a0_thru_a6: array[ 0 .. 6 ] of integer,
    status_register: 0 .. 0ffff(16),
    supervisor_stack_pointer: ↑cell,
    user_stack_pointer: ↑cell,
    program_counter: ↑cell,
    tcb_for_running_task: task_ptr,
    module_name: pmt$program_name,
    module_offset:  0 .. 0ffff(16),
    error_during_firewall: 0 .. 0ffff(16), {if non-zero then error
    firewall_procedure_address: ↑cell,
    mpb_status_register: mpb_status_word,
    case ex_error_codes of
    = bus_error_i, address_error_i =
      first_failure_capture_address: ↑cell,
      bus_exception_status: 0 .. 0ffff(16),
      access_address: ↑cell,
      instruction_register: 0 .. 0ffff(16),
    = smm_single_bit_error_i, smm_double_bit_error_i =
      smm_card_slot: 0 .. 7,
      smm_error_log: 0 .. 0ffff(16),
    casend
  recend;

TYPE
  ex_error_codes = ( unused_0,
                     unused_1,
                     bus_error_i,
                     address_error_i,
                     illegal_instruction_i,
                     zero_divide_i,
                     chk_instruction_i,
                     trapv_instruction_i,
                     privilege_violation_i,
                     trace_interrupt_i,
                     line_1010_interrupt_i,
                     line_1111_interrupt_i,
                     smm_single_bit_error_i,
                     smm_double_bit_error_i,
                     task_runs_too_long_i );

VAR
  exec_error_table: [XREF] executive_error_table;
```

FG TRIM

```
{ PROCEDURE NAME: fg_trim
{
{ PURPOSE:
{   Trim number of bytes needed from the back of the
{   data_descriptor.
{
{ CALL FORMAT:
{   (*callc CMXPFGT)
{   fg_trim(size,address,msg);
{
{ DESCRIPTION:
{   Trim from the back of the data_descriptor the number of
{   bytes needed--i.e.  size.  If a buffer is completely used
{   up, then release it from memory.  If the entire message is
{   less than size, then return false to let the caller know
{   there is not enough bytes to satisfy the request.
{

  PROCEDURE [XDCL] fg_trim (size: non_empty_message_size; {size of needed bytes
        address: ↑cell; {address of where to position bytes
      VAR msg: buf_ptr); {first data_descriptor
```

FIELD SIZE

```
{ Function Name:  field_size
{
{ Purpose:  find field size
{
{ Description:
{  This common routine converts management_data_unit field length
{  to number of bytes.
{
{ Call Format:
{       (*callc mexgdf)
{       count := field_size (len, field_type);
{
{ Returns:  zero if unsupported field type,
{           else number of bytes.

  FUNCTION [XDCL] field_size ( {
        len: 1 .. mdu_field_size;
        field_type: mdu_field_type): 0 .. mdu_field_size;
```

FILE ACCESS

```
{ PROCEDURE file_access
{
{ PURPOSE:
{   This  procedure provides the interface between the File Access User
{   and the File Access M-E.
{
{ CALL FORMAT:
{   (*callc cmxfame)
{   file_access (user_fcb);
{
{ DESCRIPTION:
{   The  file_access  procedure  is  directly called by the File Access
{   User.  The user_fcb is validated and  the  request is issued to the
{   Dependent File Access M-E which communicates  with  the  host  File
{   Server  Application  through  the Independent File Access M-E.  The
{   caller's task is put in a wait state  if  no response procedure was
{   specifed; otherwise an immediate return is made  and  the user must
{   monitor field access_complete in user_fcb.
{
{   All  file_access requests require fields request_code and response_
{   procedure  to  be  initialized  in user_fcb.  Additional fields are
{   required for some file_access requests:
{
{     open_file, create_file, delete_file:  title_name, file_name
{
{     open_file, create_file:  access_mode, access_type
{
{     write_file:  data_buffer
{
{     read_file:  read_length
{
{     seek_file:  origin, offset
{
{   Optional fields are user_id and quality (currently not used).
{
{   File_access  always  returns  fields access_complete, response_code
{   and   reject_code   in   the   user_fcb.      If  response_code  =
{   request_rejected then reject_code contains a  reason  for  failure.
{   If  any  reject_code  is  returned  on  a create_file, open_file or
{   delete_file then the file request was not  satisfied.  Reject_code
{   values  of protocol_error or unexpected_file_close are "fatal" and
{   indicate  that  the  file  is  no  longer "open"; otherwise the user
{   should  issue  a  close_file request to  clean  up  the  connection
{   through  to  the  host   file   server   application.    For other
{   reject_code  values  the  user  may try some sort of error recovery
{   algorithm.
{
{   For  response_code  =  request_confirmed  then current_position and
{   file_length (if request_code = write_file) are  updated.  If it was
{   a read_file request then data_buffer contains the  data transfered,
{   including  any  data  left  from  the  previous  read_file request.
{   (Field  line_number  is provided for the convenience  of  utilities
{   that  perform text processing  services  via  their  own  calls  to
{   file_access.)
```

```
{
{ NOTES:
{   Fields fcb, current_position, file_length, and file_server in
{   user_fcb must not be molested or unpredicatble results occur.
{
{   When using the C170 File Server, certain features apply only to
{   "writeable" files: those whose names begin with the characters
{   'dump#'.  These features are write_only and read_write access
{   mode; also these files may be created, modified, extended and
{   purged.
{
{   Otherwise, any file name may be read that is registered with the
{   C170 File Server. If the file name is not registered and request
{   code is create_file then the C170 File Server will automatically
{   register the file name (again, only if the name begins with
{   'dump#'.)
{
{   The title used with the open/create/delete file requests must
{   first be registered with Independent File Access M-E via the
{   command "define_file_support" (deffs); otherwise a file_reject
{   code of "file_service_unavailable" will be returned.  If more than
{   one host system supports the title but only one of them contains
{   the requested file, then file access will be tried to each host
{   system until the file is found.

  PROCEDURE [XDCL] file_access (
        user_fcb: ↑file_control); { file control block
```

FIND

```
{ PROCEDURE NAME: find
{
{ PURPOSE:
{   Find Table in Tree Table Access Structure.
{
{ CALL FORMAT:
{   (*callc CMXPFIN)
{   addr := find(head, key);
{
{ DESCRIPTION:
{   The tree table access structure is searched for the provided key.
{   if it is found, the associated table is returned; otherwise
{   the return is NIL. The table is returned interlocked. (i.e., task
{   pre-emption from interrupt levels is disabled.)
{
{ SEE ALSO:
{   find_copy

  PROCEDURE [XDCL] find ( {
        head: ↑root; { head root of tree
        key: integer) {key for searching operations }
    ↑ cell; {table address of associated table
```

FIND FIRST

```
{ PROCEDURE NAME: find_first
{
{ PURPOSE:
{   Find Table with Key Greater than a Given
{   Key and Return Interlocked.
{
{ CALL FORMAT:
{   (*callc CMXPFNF)
{   table = find_first(head, key, qual, param);
{
{ DESCRIPTION:
{   Locate the first entry in the tree having the stringally
{   greater key than that specified.
{   If qual <> NIL, call qual↑ (table, param, boolean_val).  and
{   return the first key having a non-zero return.  Return the
{   key in key, and return the associated table, interlocked.
{
{ SEE ALSO:
{   sfind_first
{   sfind_next
{   find_next
{
```

```
  PROCEDURE [XDCL] find_first ( {
        head: ↑root;
    VAR key: integer;
        qual: ↑procedure ( {
                        ptr: ↑cell;
                        parm_ptr: ↑cell;
                    VAR bool: boolean);
        param: ↑cell)↑ cell;
```

FIND FREE NODE

```
{ PROCEDURE NAME:   find_free_node
{
{ PURPOSE:
{   Find Free Key in Tree Structure.
{
{ CALL FORMAT:
{   (*callc CMXPFFN)
{   find_free_node(head, key_ptr);
{
{ DESCRIPTION:
{   The tree pointed to by head is searched for the occurrence
{   of the key value passed in the call.  The search is
{   performed by comparing the key to the key.numeric value in
{   the current node.
{
{   If the key.numeric at the current node is equal to the key,
{   the key is bumped by one and a check is performed to
{   determine where to continue the search from.  If the current
{   node has a right subtree the search will be continued from
{   there, otherwise the search will start over at the root.
{
{   If the key.numeric at the current node is less than the key
{   and the right subtree is NIL, then the current key value
{   will be returned.  Otherwise the search will continue down
{   the right subtree.
{
{   If the key.numeric at the current node is greater than the
{   key and the the left subtree is NIL, then the current key
{   value will be returned.  Otherwise the search will continue
{   down the left subtree of the current node.
{

  PROCEDURE [XDCL] find_free_node ( {
          head: ↑root; { pointer to root of tree.
      VAR key: integer); { pointer to key. key returned will be > key
```

FIND NEXT

```
{ PROCEDURE NAME: find_next
{
{ PURPOSE:
{   Find Table with Key Greater than a Given
{   Key and Return Interlocked.
{
{ CALL FORMAT:
{   (*callc CMXPFNX)
{   table = find_next(head, key, qual, param);
{
{ DESCRIPTION:
{   Locate the  first entry in the tree having the stringally
{   greater key than that specified.
{   If qual <> NIL, call (*qual) (table, param)
{   and return the first key having a non-zero return.
{   Return the key in key, and return the associated table,
{   interlocked.
{
{ SEE ALSO:
{   sfind_first
{   find_first
{   sfind_next

  PROCEDURE [XDCL] find_next ( {
          head: ↑root; { root of tree
      VAR key: integer; {key associated with entry - returned
          qual: ↑procedure ( {
                              ptr: ↑cell; {user specified test function
                              param_ptr: ↑cell;
                        VAR bool: boolean);
          param: ↑cell) {parameter to pass to qual
    ↑ cell; { table address of associated table
```

FIRST BYTE ADDRESS

```
{ FUNCTION NAME:  first_byte_address
{
{ PURPOSE:
{   Obtain First Byte Address of a Message.
{
{ CALL FORMAT:
{   (*callc CMXPFBA)
{   byte_address := first_byte_address (message);
{
{ DESCRIPTION:
{   This routine returns the address of the first byte of a message.
{   This is intended for fast access by protocols that especially
{   use the first byte.
```

FIRST NODE

```
{ PROCEDURE NAME: first_node
{
{ PURPOSE:
{   Generate the first node of a B-Tree.
{
{ CALL FORMAT:
{   (*callc CMXPNEW)
{   first_node(head, key, table, size)
{
{ DESCRIPTION:
{   Space is allocated for the first node on the B-Tree.  Associated
{   values are placed in the first node and the head node is linked
{   to the first node.
{

  PROCEDURE [XDCL] first_node ( {
        head: ↑root;
        key: key_record; { key for searching operations
        table: ↑node_control;
        size: integer);
```

FRAGMENT

```
{ PROCEDURE NAME:  fragment
{
{ PURPOSE:
{   Extract a Message Fragment.
{
{ CALL FORMAT:
{   (*callc CMXPFRA)
{   fragment (bytes, remainder_ptr, fragment_ptr, threshold);
{
{ DESCRIPTION:
{   The length of the message is inspected. If the specified length
{   equals or exceeds the actual length, "remainder_ptr" is
{   set to NIL and the entire message is returned in "fragment_ptr".
{
{   Otherwise, the portion of the message to be removed is inspected
{   for multiple ownership. If this case is found, the multiply owned
{   portion is logically copied and released.
{
{   The portion to be removed is then removed from the message. If
{   it terminates on other than an even buffer boundary, the affected
{   buffer is logically copied, and the copy is appended to the
{   fragment.
{
{   Upon return, "remainder_ptr" addresses the descriptor of the
{   remaining portion of the message, rather than the start of the
{   message.
{
{ SEE ALSO:
{   Assemble

  PROCEDURE [XDCL] fragment ( {
          size: non_empty_message_size; { nr. of bytes to include in fragment
      VAR remainder_ptr: buf_ptr;       { address of message buffer
      VAR fragment_ptr: buf_ptr;        { address of message fragment buffer
          threshold: threshold_size);   { threshold for buffer acquisition
```

GENERIC TRANSPORT INTERFACE DEFINITIONS

```
{  TABLE NAME:  Generic Transport Interface Definitions
{
{  DECK NAME:  TRDGT
{

  CONST
    gt_layer_mgmt_title = 'generic_transport',
    gt_max_credit_window = 8;


  TYPE
    gt_status =     (gt_request_processed,
                     gt_credit_not_within_limits,
                     gt_source_sap_not_found,
                     gt_message_exceeds_max_length,
                     gt_invalid_state,
                     gt_sap_open,
                     gt_sap_busy,
                     gt_no_memory_for_sap,
                     gt_connection_not_found,
                     gt_no_memory_for_connection),

    gt_credit_window_range = 1 .. gt_max_credit_window;


  TYPE
    gt_connection_mgmt_call = ↑procedure ( {
      VAR request: gt_connection_mgmt_request);

  TYPE
    gt_layer_mgmt_call = ↑procedure ( {
      VAR request: gt_layer_mgmt_request);

  TYPE
    gt_layer_mgmt_codes = ( {
      gt_open_sap, gt_close_sap, gt_connect_request);


  TYPE
    gt_layer_mgmt_request = record
      workcode: gt_layer_mgmt_codes, {request primitive to be processed ;input
      service_sapid : gt_sap, { source sap  ; input or output
      status : gt_status, {status of the request ; output

      case gt_layer_mgmt_codes of

      = gt_open_sap =
        open_sap@: record
          user_sapid: ↑cell, {transport user's sapid ; input
          dedicated_sapid: sap_id_type, {dedicated sap  ;input
          user_layer_mgmt_if: generic_connect_if, {layer management indication ;input
```

```
      user_connect_mgmt_if: generic_data_if, {connection mgm  indication ;input
      generic_connect_mgmt_if: gt_connection_mgmt_call,
            {transport procedure for connection mgmt ; output
    recend,

  = gt_connect_request =
    connect_request@: record
      user_cepid: ↑cell, {transport user's cepid ; input
      destination: gt_sap, {destination address ;input
      credit_window: gt_credit_window_range, { variabe window size ;input
      connect_data: buf_ptr, { user data  ;input
      priority: generic_priority,  {priority of the connection ;input
      service_cepid: ↑cell, {generic transport cepid ;output
    recend,

  = gt_close_sap =

  casend,
recend;


TYPE
  gt_connection_mgmt_codes = ( {
    gt_connect_accept, gt_data_request, gt_xdata_request, gt_disconnect_request,
    gt_flow_control_request, gt_abort_request);


TYPE
  gt_flow_control_request_code = (gt_start_request, gt_stop_request);

TYPE
  gt_connection_mgmt_request = record
    workcode: gt_connection_mgmt_codes,{request primitive to be processed ;input
    service_cepid: ↑cell, {generic_transport cepid  ;input
    status: gt_status, {status of request ;output

    case gt_connection_mgmt_codes of

    = gt_connect_accept =
      connect_accept@: record
        priority: generic_priority, {connection priority ;input
        credit_window: gt_credit_window_range,{window size  ;input
        accept_data: buf_ptr, {user data  ;input
      recend,

    = gt_data_request, gt_xdata_request, gt_disconnect_request =
      user_data: buf_ptr, {data passed with the request ;input

    = gt_flow_control_request =
      flow_control_code : gt_flow_control_request_code, {start or stop request ;input

    = gt_abort_request =

    casend,
```

```
    recend;


{
{  INDGENERIC
{
{       indication codes presented by generic Transport to user
{       interface routines.

  TYPE
    indgeneric = (connect_indication, connect_confirm, disconnect_indication,
      data__indication, xdata_indication, start_indication, stop_indication,
      start_xdata_indication, stop_xdata_indication);

{
{ Connect Indication Interface Routine Type
{
{      This interface must be used for the Connect  Indication
{ Interface Routine.
{

  TYPE
    generic_connect_if = ↑procedure (cepid: generic_cepid;
      VAR sdu: buffer;
      source: gt_sap;
      user_sap: usapid;
      VAR cepid: ucepid);

{
{ Generic Data Delivery Interface
{
{      This interface routine is used for all indications  and
{ confirmations other than the Connect Indication.
{

  TYPE
    generic_data_if = ↑procedure (interface: indgeneric;
      cepid: ucepid;
      VAR sdu: buffer);

{
{ Connection Priority
{
{      Connections are serviced at different priorities  based
{ on data path type.
{

  TYPE
    generic_priority = (low, high);


{
```

```
{ Generic CEPID
{
{      This data element must be passed to Generic request and
{ response   interfaces   to   identify   the   connection   being
{ operated on.
{

  TYPE
    generic_cepid = ↑cell;

{
{ User Cepid
{     this data element  is the user's CEP idneitfication
{

  TYPE
    ucepid = ↑cell;

{
{ User SAP Identifier
{
{      This value is presented to the user on Connect  Indica-
{ tions to identify the SAP to the user.
{

  TYPE
    usapid = ↑cell;

*callc trdsap
```

GEN DATA FIELD

```
{ Procedure Name:  gen_data_field
{
{ Purpose: generate data field in management data units
{
{ DESCRIPTION:
{  This function is the reverse of get_data_field.  It generates a
{  data field appended to the message as specified by the data field
{  type.  Buffers will be appended as necessary.
{
{ Call Format:
{       (*callc mexgdf)
{       gen_data_field (msgbuf, field_cell, len, type);
{
{ Entry Conditions
{  If msgbuf = NIL, no first buffer exists and one is gotten.
{
{ Exit Conditions
{  msgbuf = message in management data unit syntax
{
{ Limitations:
{  Compressed data fields are not generated.
{  Unsupported data field types or a bad length will not generate a field.

  PROCEDURE [XDCL] gen_data_field ( {
      VAR msgbuf: buf_ptr; { ptr to buffer containing data field(s)
          field_cell: ^cell; { data field
          len: 1 .. mdu_field_size; { data field length
          typ: mdu_field_type); { data field type
```

GEN TEMPLATE ID

```
{ Procedure Name:  gen_template_id
{
{ Purpose: generate a template identifier
{
{ DESCRIPTION:
{  This function places the specified template identifier in the message
{  (buffer) provided.  The template identifier is used to identify the
{  template associated with the message.  The message consists of variable
{  information to be combined with the associated template.  Each variable
{  part of the message is placed in the message buffer via the common
{  subroutine gen_data_field.
{
{ Call Format:
{       (*callc csxgti)
{       gen_template_id (msgbuf, template_id);
{
{ Entry Conditions
{  If msgbuf = NIL, no first buffer exists and one is gotten.
{
{ Exit Conditions
{  msgbuf = message in management data unit syntax
{
{ Limitations:
{  Compressed data fields are not generated.
{  Unsupported data field types or a bad length will not generate a field.

  PROCEDURE [XDCL] gen_template_id ( {
      VAR msgbuf: buf_ptr; { ptr to buffer containing data field(s)
          template_id: template_id_type);
```

GET CARD TYPE AND ADDRESS

```
{  PROCEDURE NAME: get_card_type_and_address
{
{  PURPOSE:
{    The  purpose of this procedure is to get the card type and
{    card address for the device name specified.
{
{  CALL FORMAT:
{    (*callc sdxgcta)
{    get_card_type_and_address   (device_name,   device_record,
{    device_available)
{
{  DESCRIPTION:
{    The device name provided is parsed to determine its
{    validity.  If a valid device name was specified and the
{    associated board type is physically available in the
{    associated System Status Table, then the device card type
{    and card address is returned along with a successful
{    status indication.  Otherwise, a status indication is
{    returned which indicates that the device name is not
{    available in the DI.
{
{    Parameter    Description
{    device_name
{                 This  parameter  identifies  the hardware
{                 device  name  whose card type and address is
{                 desired.
{
{    device_record
{                 This  is  a  return  parameter which contains
{                 the  card  type  and  card  address  for  the
{                 device name specified.
{
{    device_available
{                 This  is  a  return parameter which indicates
{                 if the device name specified is in the DI. If
{                 the  device  name  is  in  the  DI  TRUE  is
{                 returned; otherwise, FALSE is returned.
{
{  GLOBAL DATA REFERENCED:
{    major_card_status_table
{    lim_status_table
{    port_status_table(s)
{    smm_bank_status_table(s)
{    pmm_bank_status_table


  PROCEDURE [XDCL] get_card_type_and_address ( {
        device_name: string (*);
    VAR device_record: card_info_record;
    VAR device_available: boolean);
```

GET COMMAND LINE

```
{ PROCEDURE NAME:  get_command_line
{
{ PURPOSE:
{   Read a Procedure File, and pass the SCL command lines one by
{   one to the caller.
{
{  CALL FORMAT:
{      (*callc cmxgcl)
{      get_command_line (user_fcb, command, read_status);
{
{ DESCRIPTION:
{   The next command line is read from a file and delivered in an
{   edited form.  get_command_line will:
{     - compress multiple blanks down to a single (except for strings)
{     - replace comments by a single blank
{     - process elipsis (.. at the end of a line)
{     - remove leading and trailing blanks
{     - remove totally empty lines (after above processing is done)
{     - maintain a running line number counter (set to 0 by caller)
{   A unit separator or an end of file terminates a line (regardless
{   where encountered).
{   Processed data buffers are released.
{
{ NOTES:
{   The caller must OPEN the file prior to calling this procedure
{   (the caller should not issue any READ's) via an open_file
{   request to the file_access procedure.

  PROCEDURE [XDCL] get_command_line ( {
        fcb: ↑file_control;
    VAR command: ost$string;
    VAR read_status: read_file_status);
```

GET DATA FIELD

```
{ Procedure Name:  get_data_field
{
{ Purpose:  get data field from management data units
{
{ DESCRIPTION:
{  This common function extracts a data field from management data unit
{  formatted messages and returns it in an internal format.  The buffer
{  pointer is updated as fields are extracted.  The memory extent gotten
{  for the data field must be returned by the caller.
{
{  Since a data field may consist of several sub-fields, the data is
{  previewed to determine how much memory is needed.  Then the fields
{  are stripped until the field-complete flag is seen.
{
{ Call Format:
{       (*callc mexgdf)
{       get_data_field (msgbuf, field_cell, len, type);
{
{ Exit Conditions
{ returns:  address of extracted field or NIL
{           if no more data is available or if any errors
{           are encountered in the data fields.
{ msgbuf: updated as necessary.
{
{ Cautions
{ Caller is responsible for FREEing the memory extent gotten.

  PROCEDURE [XDCL] get_data_field ( {
      VAR msgbuf: buf_ptr; { ptr to buffer containing data unit
      VAR field_cell: ↑cell; { returned data field
      VAR len: 0 .. mdu_field_size; { returned data field length
      VAR typ: mdu_field_type); { returned data field type
```

GET DATA LINE

```
{ PROCEDURE NAME:  get_data_line
{
{ PURPOSE:
{   Read a text file, and pass the data lines one by one to the caller.
{
{  CALL FORMAT:
{     (*callc cmxgdl)
{     get_data_line (user_fcb, line, read_status);
{
{ DESCRIPTION:
{   The next data line is read from a file and delivered in a string.
{   A unit separator, an end of file (regardless where encountered) or
{   a maximum of ost$max_string_size characters (if no unit seperator
{   or end of file encountered) terminates a line.
{   Processed data buffers are released.
{
{ NOTES:
{   The caller must OPEN the file prior to calling this procedure
{   (the caller should not issue any READ's) via an open_file
{   request to the file_access procedure.

  PROCEDURE [XDCL] get_data_line ( {
        fcb: ↑file_control;
    VAR line: ost$string;
    VAR read_status: read_file_status);
```

GET EXPRESS,

```
{ PROCEDURE NAME:  get_express,
{                  maybe_express
{
{ PURPOSE:
{   Get Intertask Message from Express Queue.
{
{ CALL FORMAT:
{   (*callc CMXMTSK)
{   get_express (address, sender);
{   maybe_express (address, sender);
{
{ DESCRIPTION:
{   If a message is found on the express queue,
{   it is copied to the addressed space, and removed from the
{   intertask message queue.  The normal queue is not inspected.
{   The following calls have the following effects :
{
{   NAME:            TRAP NUMBER:     EFFECTS:
{   get_express           1           control returns after a message
{                                     has been made available to the
{                                     caller.
{
{   maybe_express         0           a message is obtained,
{                                     or a failure is returned.

  PROCEDURE [INLINE] get_express ( {
        intertask_message: ↑cell;
    VAR task_sending_message: task_ptr);
```

GET FIRST BYTE

```
{ FUNCTION NAME:  get_first_byte
{
{ PURPOSE:
{   Obtain First Byte of Message Text.
{
{ CALL FORMAT:
{   (*callc CMXPGFB)
{   byte := get_first_byte (message);
{
{ DESCRIPTION:
{   This routine returns the first valid text byte of a message.
{   This is intended for fast access by protocols that especially
{   use the first byte.


  FUNCTION [INLINE] get_first_byte (d: buf_ptr): char;
    get_first_byte := d↑.the_data↑.data_text (d↑.offset);
```

GET LAST BYTE

```
{ FUNCTION NAME:  get_last_byte
{
{ PURPOSE:
{   Get last byte from a given descriptor.
{
{ DESCRIPTION:
{   This function locates the last byte of the given buffer chain
{   and returns that byte to the caller (as a character).
{
{ NOTES:
{   It is assumed that the last buffer in the chain will not be empty.
{

  FUNCTION [INLINE] get_last_byte ( {
        first_descriptor: buf_ptr): char;
```

GET LONG BUFFERS

```
{ PROCEDURE NAME:  get_long_buffers
{
{ PURPOSE:
{   Get One or More Data Buffers.
{
{ CALL FORMAT:
{   (*callc CMXPGBF)
{   buffer_address := get_long_buffers (count, buffer_address, threshold);
{   buffer_address := fg_long_buffers (count, buffer_address, threshold);
{   buffer_address := maybe_long_buffers (count, buffer_address, threshold);
{
{ DESCRIPTION:
{ The executive function Get Data Buffer Chain (4.3) is called,
{   with the following entry type:
{
{    NAME:                TRAP NUMBER:    EFFECTS:
{
{    get_long_buffers          1          the buffers are obtained
{
{    fg_long_buffers           2          interrupt routine use only;
{                                         the buffers are obtained
{                                         or a failure is returned.
{
{    maybe_long_buffers        0          the buffers are obtained
{                                         or a failure is returned.

   PROCEDURE [XDCL] get_long_buffers ( {
         number_of_buffers: buffer_request_limit;
     VAR buffer_chain_allocated: buf_ptr;
         threshold_index: threshold_size);
```

GET MEMORY

```
{ PROCEDURE NAME:  get_memory
{
{ PURPOSE:
{   Get Global Memory Extent.
{
{ CALL FORMAT:
{   (*callc CMXPGGX)
{   get_memory (address, size);
{   fg_memory (address, size);
{   maybe_memory (address, size);
{
{ DESCRIPTION:
{   The executive function Get Global Memory Extent (4.5) is called,
{   with the following entry type:
{
{   NAME:                  TRAP NUMBER:    EFFECTS:
{
{   get_memory                  1          the memory extent is obtained
{
{   fg_memory                   2          interrupt routine use only;
{                                          the memory extent is obtained
{                                          or a failure is returned.
{
{   maybe_memory                0          the memory extent is obtained
{                                          or a failure is returned.

    PROCEDURE [XDCL] get_memory ( {
      VAR extent_returned: ↑cell;
          extent_size: executive_extent);
```

GET MESSAGE LENGTH

```
{ FUNCTION NAME:  get_message_length
{
{ PURPOSE:
{   Get Message Length.
{
{ CALL FORMAT:
{   (*callc CMXPGML)
{   size := get_message_length (buffer_address);
{
{ DESCRIPTION:
{   The number of bytes in the message is returned.
{

  FUNCTION [INLINE] get_message_length (message: buf_ptr): message_size;
```

GET MPB EXTENT

```
{ PROCEDURE NAME:  get_mpb_extent
{
{ PURPOSE:
{   Get MPB RAM Memory Extent.
{
{ CALL FORMAT:
{   (*callc CMXPGMP)
{   get_mpb_extent (address, size);
{   fg_mpb_extent (address, size);
{   maybe_mpb_extent (address, size);
{
{ DESCRIPTION:
{   The executive function Get MPB RAM Memory Extent is called,
{   with the following entry type:
{
{   NAME:                TRAP NUMBER:    EFFECTS:
{
{   get_mpb_extent           1           the memory extent is obtained
{
{   fg_mpb_extent            2           interrupt routine use only;
{                                        the memory extent is obtained
{                                        or a failure is returned.
{
{   maybe_mpb_extent         0           the memory extent is obtained
{                                        or a failure is returned.


    PROCEDURE [XDCL] get_mpb_extent ( {
      VAR extent_returned: ↑cell;
          extent_size: executive_extent);
```

GET MSG

```
{ PROCEDURE NAME:  get_msg
{                  maybe_msg
{
{ PURPOSE:
{   Get Intertask Message from Normal or Express Queue.
{
{ CALL FORMAT:
{   (*callc CMXMTSK)
{   get_msg (address, sender);
{   maybe_msg (address, sender);
{
{ DESCRIPTION:
{   If a message is found on either the normal or express queue,
{   it is copied to the addressed space, and removed from the
{   intertask message queue.  The express queue has priority.
{   The following calls have the following effects :
{
{   NAME:            TRAP NUMBER:     EFFECTS:
{   get_msg               1           control returns after a message
{                                     has been made available to the
{                                     caller.
{
{   maybe_msg             0           a message is obtained,
{                                     or a failure is returned.
{
```

```
  PROCEDURE [INLINE] get_msg ( {

      intertask_message: ↑cell;
    VAR   task_sending_message: task_ptr);
```

GET NEXT STATUS SAP

```
{ PROCEDURE:  get_next_status_sap
{
{ PURPOSE:
{   The  purpose  of this procedure is to provide a command
{   processor the ability  to  retrieve  the address of its
{   associated  software  components  status  tables   when
{   multiple copies are executing at the same time.
{
{ CALL FORMAT:
{   (*callc sdxssar)
{   get_next_status_sap      (name,     last_sap_table_ptr,
{   next_sap_table_ptr, task_id, successful, response)
{
{ DESCRIPTION:
{   If  multiple  copies  of  a  software  component can be
{   executing  at  the  same  time  in  the  DI  then   the
{   get_next_status_sap  common  subroutine must be used by
{   the  software  components  associated   status  command
{   processor  to retrieve the status table for  each  copy
{   executing.    This   mechanism   allows   the   command
{   processor  to  be  in  a  different  module  than  the
{   software component(s)  and  thus can be invoked without
{   loading the associated software  component if it is NOT
{   already  loaded  in  the  DI.  If  the  value  of  the
{   parameter   last_sap_table_ptr   is   NIL   then   the
{   next_status_table_ptr parameter  returned contains the
{   address  of  the first associated status  sap  for  that
{   software component.  The software component can then
{   get  the  next  associated status sap in the  table  by
{   calling  this  routine  again  with  the  parameter
{   last_sap_table_ptr  set  to  the returned value of  the
{   parameter next_sap_table_ptr from  the  previous  call.
{   When  all the associated saps have been retrieved, then
{   NIL  will  be  returned  in  the  return  parameter
{   next_sap_table_ptr.     If    the    last_sap_table_ptr
{   provided  on  the  call  can not be found then the
{   parameter  successful  is  returned as FALSE.  If  the
{   status_table_ptr  is  returned  NIL then either the
{   software component has not opened  a  status  sap or it
{   has  no  status to report.  In either case the response
{   parameter returned  contains  the  appropriate response
{   and must be returned to the origin of  the  command via
{   the  Dependent  Command M-E.  If the software component
{   opened a sap but has  no  associated  status table then
{   the following response is returned:
{
{      Software component "name" loaded.
{
{   If  the  software component has not opened a status sap
{   then the following response is returned:
{
{      Software component "name" not registered to report status.
```

```
{
{   Parameter Description
{   name: (input)
{       This parameter is the name of the software component.
{
{   last_sap_table_ptr: (input)
{       This  parameter  identifies the address of the software
{       components status table  of the previously obtained sap
{       from the table.  If no previous  sap  was obtained then
{       NIL should be passed.
{
{   next_sap_table_ptr: (output)
{       This  parameter  identifies the address of the software
{       components status table  of  the next registered sap in
{       the table.
{
{   task_id: (output)
{       This  parameter  identifies the task_id of the software
{       component who opened the software status sap.
{
{   successful: (output)
{       This   parameter   is   returned   as   TRUE   if   the
{       last_sap_table_ptr   was   found;   otherwise,   it   is
{       returned as FALSE.
{
{   response: (output)
{       This  is  a  return parameter which contains a response
{       to be sent to the  origin  of  the command if its value
{       is not NIL.
{
{ GLOBAL DATA REFERENCED:
{   software_status_sap_table
{
{ NOTES AND CAUTIONS:
{   The    procedure   NOPREMPT   is   called   upon   entering
{   get_next_status_sap    to    suppress    task   preemption.
{   Get_next_status_sap  is  exited in a non-preemptable  state
{   and  will  require  the  caller  to  make  a  call  to  the
{   procedure OKPREMPT if preemptability is so desired.


  PROCEDURE [XDCL] get_next_status_sap ( {
        name: string ( * <= 31);
        last_sap_table_ptr: ↑cell;
    VAR next_sap_table_ptr: ↑cell;
    VAR task_id: task_ptr;
    VAR successful: boolean;
    VAR response: buf_ptr);
```

GET PMM EXTENT

```
{ PROCEDURE NAME:  get_pmm_extent
{
{ PURPOSE:
{   Get Private Memory Extent.
{
{ CALL FORMAT:
{   (*callc CMXPGPM)
{   get_pmm_extent (address, size);
{   fg_pmm_extent (address, size);
{   maybe_pmm_extent (address, size);
{
{ DESCRIPTION:
{   The executive function Get Private Memory Extent (4.6) is called,
{   with the following entry type:
{
{   NAME:                TRAP NUMBER:    EFFECTS:
{
{   get_pmm_extent            1          the memory extent is obtained
{
{   fg_pmm_extent             2          interrupt routine use only;
{                                        the memory extent is obtained
{                                        or a failure is returned.
{
{   maybe_pmm_extent          0          the memory extent is obtained
{                                        or a failure is returned.

    PROCEDURE [XDCL] get_pmm_extent ( {
      VAR extent_returned: ↑cell;
          extent_size: executive_extent);
```

GET SHORT BUFFERS

```
{ PROCEDURE NAME:  get_short_buffers
{
{ PURPOSE:
{   Get One or More Descriptor Buffers.
{
{ CALL FORMAT:
{   (*callc CMXPGDB)
{   buffer_address := get_short_buffers (count, buffer_address, threshold);
{   buffer_address := fg_short_buffers (count, buffer_address, threshold);
{   buffer_address := maybe_short_buffers (count, buffer_address, threshold);
{
{ DESCRIPTION:
{   The executive function Get Descriptor Buffer Chain (4.1) is called,
{   with the following entry type:
{
{   NAME:                 TRAP NUMBER:    EFFECTS:
{
{   get_short_buffers          1          the buffers are obtained
{
{   fg_short_buffers           2          interrupt routine use only;
{                                         the buffers are obtained
{                                         or a failure is returned.
{
{   maybe_short_buffers        0          the buffers are obtained
{                                         or a failure is returned.

    PROCEDURE [XDCL] get_short_buffers ( {
          number_of_buffers: buffer_request_limit;  948
      VAR buffer_chain_allocated: buf_ptr;  954
          threshold_index: threshold_size);  27
```

GET SIZE N ADDR

```
{ PROCEDURE NAME:  get_size_n_addr
{
{ PURPOSE:
{   Get size and address of memory extent for section
{
{ CALLING FORMAT:
{   *callc sixgsiz
{   get_size_n_addr (section_address, section_size);
{
{ DESCRIPTION:
{   The memory extent size and address are determined from the indicated
{ start section address.
{

  PROCEDURE [XREF] get_size_n_addr ( {
    VAR section_address: ↑cell;
    VAR section_size: dlt$section_length);
```

GET SOURCE ADDRESS

```
{ Procedure Name:  get_source_address
{
{ Purpose:  get command source address
{
{ Description:
{  This routine gets the command source address from the
{  Command M-E command/response table.
{
{ Call Format:
{       (*callc mexgsa)
{       get_source_address (source, task);
{
{ Entry Conditions
{  task = Command Processor I/F task_ptr or
{         NIL if current task_ptr is to be used
{
{ Exit Conditions
{  source = 0 is returned if the command/response table cannot be found.

  PROCEDURE [XDCL, #GATE] get_source_address ( {
    VAR source: generic_sap;
    VAR task: task_ptr); { Command Processor I/F task_ptr
```

GET STATUS RECORD

```
{  PROCEDURE NAME: get_status_record
{
{  PURPOSE:
{    The purpose of this procedure is retrieve a status record for
{    the device name specified.
{
{  CALL FORMAT:
{    (*callc sdxgpsr)
{    get_status_record (device_name, device_status_record,
{          device_available)
{
{  DESCRIPTION:
{    The device name provided is parsed to determine its validity.
{    If a valid device name was specified and the associated board type
{    is physically available in the associated System Status Table, then
{    the device status record is returned along with a successful status
{    indication.  Otherwise, a status indication is returned which
{    indicates that the device name is not available in the DI.
{
{    Parameter     Description
{    device_name   This parameter identifies the hardware device name
{                  whose status record is desired.
{
{    device_status_record  This is a return parameter which contains
{                  the status record for the device name specified.
{
{    device_available  This is a return parameter which indicates if the
{                  device name specified in the DI.  If the device name
{                  is in the DI TRUE is returned; otherwise, FALSE is
{                  returned.
{
{  GLOBAL DATA REFERENCED:
{    major_card_status_table
{    lim_status_table
{    port_status_table(s)
{    smm_bank_status_table(s)
{    pmm_bank_status_table


  PROCEDURE [XDCL] get_status_record ( {
        device_name: string (maximum_device_name_size);
    VAR device_status_record: component_status_type;
    VAR device_available: boolean);
```

GET STATUS SAP

```
{ PROCEDURE:  get_status_sap
{
{ PURPOSE:
{   The  purpose  of this procedure is to provide a command
{   processor the ability  to  retrieve  the address of its
{   associated software component status table.
{
{ CALL FORMAT:
{       (*callc sdxssar)
{       get_status_sap    (name,   sap_table_ptr,   task_id,
{       response)
{
{ DESCRIPTION:
{   The  command  processor  responsible for generating the
{   status  of  a  particular software  component  utilizes
{   this  subroutine  to  retrieve  the  address  of  the
{   software  components  status  tables.  This  mechanism
{   allows  the  command processor to  be  in  a  different
{   module than the  software  component  and  thus  can be
{   invoked  without  loading  the  associated  software
{   component if it is NOT already  loaded  in  the DI.  If
{   the  software  component  has  opened  a  sap  then the
{   status_table_ptr  parameter  returned  contains  the
{   address of the associated status table and  the command
{   processor  can  generate  the  appropriate  status
{   response.  If  the  status_table_ptr  is  returned NIL
{   then  either  the  software component has not opened  a
{   status sap or it has  no  status  to report.  In either
{   case  the  response  parameter  returned  contains  the
{   appropriate  response  and  must  be  returned  to  the
{   origin  of the command via the Dependent  Command  M-E.
{   If the software  component  opened  a  sap  but  has no
{   associated status table then the following response  is
{   returned:
{
{      Software component "name" loaded.
{
{   If  the  software component has not opened a status sap
{   then the following response is returned:
{
{      Software component "name" not registered to report status.
{
{   Parameter Description
{   name: (input)
{       This  parameter  is  the  object  name  of the software
{       component.
{
{   sap_table_ptr: (output)
{       This  parameter  identifies the address of the software
{       components status table.
{
{   task_id: (output)
```

```
{       This  parameter  identifies the task_id of the software
{       component who opened the software status sap.
{
{   response: (output)
{       This  is  a  return parameter which contains a response
{       to be sent to the  origin  of  the command if its value
{       is not NIL.
{
{ GLOBAL DATA REFERENCED:
{   software_status_sap_table
{
{ NOTES AND CAUTIONS:
{   The    procedure   NOPREMPT   is   called   upon   entering
{   get_status_sap     to     suppress     task     preemption.
{   Get_status_sap  is  exited  in a non-preemptable state  and
{   will require the caller to make  a  call  to  the procedure
{   OKPREMPT if preemptability is so desired.


  PROCEDURE [XDCL] get_status_sap ( {
        name: string ( * <= 31);
    VAR sap_table_ptr: ↑cell;
    VAR task_id: task_ptr;
    VAR response: buf_ptr);
```

GROW

```
{ PROCEDURE NAME: grow
{
{ PURPOSE:
{   Add New Table to Tree Table Access Structure.
{
{ CALL FORMAT:
{   (*callc CMXPGRO)
{    addr :=grow(head, key, table, size)
{
{ DESCRIPTION:
{   The tree is searched for an existing association between the
{   provided key and a table structure.  If such a one exists,
{   the associated table is returned, and no update is performed.
{   Otherwise, such an association is created, and NIL is returned.
{   The table is returned interlocked (i.e. task pre-emption from
{   interrupt levels is disabled.)
{

  PROCEDURE [XDCL] grow ( {
          head: ↑root; { root of the tree
          key: integer; { key for searching operations
          t: ↑cell; { table to be added to the tree
          size: integer)↑ cell;
```

INCREMENT MODULE USE COUNT

```
{ PROCEDURE NAME:  increment_module_use_count
{
{ PURPOSE:
{   increment the module use count
{
{ CALL FORMAT:
{   *callc dlximuc
{   increment_module_use_count(entry_point_name, entry_point_found);
{
{ DESCRIPTION:
{   The module use count of the indicated entry point is incremented to
{ prevent module deloading.  If the given entry point name is all blanks,
{ then the module use count of the first module of the currently
{ running task is incremented.  The module must already be loaded.
{ This procedure is only used when one of the procedures: start_named_task,
{ load_entry_point or load_absolute_module has not been used to prevent
{ module deloading.
{

  PROCEDURE [XDCL] increment_module_use_count
    ({
         entry_point_name: pmt$program_name;
     VAR entry_point_found: boolean);
```

INIT ROOT

```
{ PROCEDURE NAME:  init_root
{
{ PURPOSE:
{   Initialize Root of Tree.
{
{ CALL FORMAT:
{   (*callc CMIPINT)
{   init_root (root, type_node, dump_id);
{
{ DESCRIPTION:
{   The root of a tree is initialized.  This includes setting up
{   initial values for interlocks and node addresses, as well as
{   setting up the (up to) four character ASCII name of the
{   table stored in each node.
{

  PROCEDURE [INLINE] init_root ( {
        r: ↑root;
        t: key_type;
        n: string (4));


    r↑.num_tables := 0; { number of tables in tree
    r↑.num_nodes := 0; { total number of nodes in the tree
    r↑.link := NIL;
    r↑.type_node := t;
    r↑.dump_id := n;
  PROCEND init_root;
```

INTERTASK MESSAGE WORKCODE DEFINITIONS

```
{  TABLE NAME:  intertask message workcode definitions
{
{  DECK NAME:  CMDITM
{

            {================================================}
            { EXECUTIVE INTERTASK MESSAGE WORKCODE DEFINITIONS }
            {================================================}

   ,exec_iptfail1                = 0000(16) { Bus/address error in interrupt
   ,exec_iptfail2                = 0001(16) { Other error in interrupt
   ,exec_tskfail1                = 0002(16) { Bus/address error in task
   ,exec_tskfail2                = 0003(16) { Other error in task
   ,exec_unused                  = 0004(16) { Reserved for Executive
   ,exec_stoptask                = 0005(16) { Stop Task
   ,exec_aborttask               = 0006(16) { Abort Task
   ,exec_new_vector_owner        = 0007(16) { New vector owner
   ,exec_unused_1                = 0008(16) { Reserved for Executive
   ,exec_dest_failed             = 0009(16) { Destination failed
   ,exec_too_much_time           = 000a(16) { Excess Slice
   ,exec_error                   = 000b(16) { MPB failure error for system_ancestor
   ,exec_end_of_day              = 000c(16) { End of day message to timer task
   ,exec_new_time                = 000d(16) { New time of day request for timer task
   ,exec_periodic_timer          = 000e(16) { Periodic timer request for timer task
   ,exec_after_interval          = 000f(16) { After interval timer request for timer t
   ,exec_at_time                 = 0010(16) { Call at time request for timer task
   ,exec_periodic_after          = 0011(16) { Periodic request after interval for tim

          {================================================}
          { COMMAND ME INTERTASK MESSAGE WORKCODE DEFINITIONS }
          {================================================}

   ,c_me_msgcode                 = 0014(16) { Command Processor I/F task
   ,c_me_respcode                = 0015(16) { Response to clp_process_command
   ,c_me_xport_msg               = 0016(16) { Command from transport I/F
   ,c_me_3b_msg                  = 0017(16) { Command from internet I/F
   ,c_me_cp_task_abort           = 0018(16) { Command processor abort
   ,c_me_cp_task_stop            = 0019(16) { Command processor stopped
   ,c_me_command_err             = 001A(16) { Command-ME processing error
   ,c_me_load_cmd                = 001B(16) { Load command processor

          {================================================}
          { ROUTING ME INTERTASK MESSAGE WORKCODE DEFINITIONS }
          {================================================}

   ,r_me_full_update_lcrds       = 0030(16) { Update Least Cost Routing Data Store
   ,r_me_part_update_lcrds       = 0031(16) { Partial update to LCRDS
   ,r_me_ridu_msg                = 0032(16) { Routing Information Data Unit message
   ,r_me_3a_nw_update            = 0033(16) { Routing 3A Network Update message
   ,r_me_periodic_ridu_process   = 0034(16) { Routing LDCNDS/RIDU process

          {================================================}
```

```
        { ERROR ME INTERTASK MESSAGE WORKCODE DEFINITIONS }
        {=================================================}

,err_me_internet_error          = 0039(16) { Internet error message

                {================================}
                { INDEPENDENT FILE ACCESS M.E.   }
                {================================}

,ifa_C170_boot_workcode         = 0040(16) { independent file access initialization
,ifa_deffs_cmd_workcode         = 0041(16) { message received from define_file_support cmd
,ifa_svm_cc_ind                 = 0042(16) { svm call confirm indication
,ifa_bip_ind                    = 0043(16) { bip indication
,ifa_xport_connect_ind          = 0044(16) { transport connect indication
,ifa_xport_data_ind             = 0045(16) { transport data indication
,ifa_canfs_cmd_workcode         = 0046(16) { transport data indication
,ifa_timeout_workcode           = 0047(16) { transport data indication

                {================================}
                { CONSOLE DRIVER WORKCODE DEFS   }
                {================================}

,console$traffic                = 0050(16) { Transmit message
,console$configuration          = 0051(16) { Startup configuration
,console$write_complete         = 0052(16) { Completion of transmission sequence
,console$read_complete          = 0053(16) { Message has been received
,console$read_correct           = 0054(16) { Message received for editing

                {================================}
                { ONLINE LOADER  WORKCODE DEFS   }
                {================================}

,dlc$load_abs_delay             = 0060(16) { load absolute module
,dlc$load_abs_proceed           = 0061(16) {
,dlc$load_entry_point_delay     = 0062(16) { load relocatable module
,dlc$load_entry_point_proceed   = 0063(16) {
,dlc$start_task_delay           = 0064(16) { load relocatable module and
,dlc$start_task_proceed         = 0065(16) { initialize as a task
,dlc$load_module_for_retain     = 0066(16) { load module
,dlc$load_cmd_proc_delay        = 0067(16) { load a command_processor
,dlc$load_cmd_proc_proceed      = 0068(16)

            {=======================================}
            {  DVM itm command and response constants }
            {=======================================}

,dvm_response_base              = 0100(16) { offset for dvm responses
,dvm_line_configure_res         = 0101(16) { line configured status
,dvm_line_reconfigure_res       = 0102(16) { line configuration response
,dvm_line_delete_res            = 0103(16) { delete line response
,dvm_line_enable_res            = 0104(16) { line enabled response
,dvm_line_disable_res           = 0105(16) { line disabled response
,dvm_data_input_res             = 0106(16) { input response
,dvm_data_output_res            = 0107(16) { output response
```

```
,dvm_terminate_io_res           = 0108(16) { line terminatio  response
,dvm_line_status_res            = 0120(16) { line status response
,dvm_trap_res                   = 0121(16) { dvm trap occurred
,dvm_timer_expired              = 0122(16) { dvm heart beat timer expired
,dvm_line_suspended             = 0123(16) { dvm has suspended service to an ip
,dvm_line_resumed               = 0124(16) { dvm has resumed previously suspended se
,dvm_line_terminated            = 0125(16) { service to a line has been terminated
,dvm_ip_dead                    = 0126(16) { intelligent peripheral has reported dea
,dvm_restart_ip                 = 0127(16) { request restart IP service
,dvm_abort_ip                   = 0128(16) { request abort IP service
,dvm_unexpected_interrupt       = 0129(16) { unexpected interrupt
,dvm_discarded_status           = 0130(16) { status discarded from queue

          {=======================================}
          { HDLC itm command and response constants }
          {=======================================}

,hdlc_command_base              = 0200(16) { HDLC ssr command base
,hdlc_wake_up_cmd               = 0201(16) { NOP message to wake up SSR (used by 3A)

,hdlc_i_timeout_cmd             = 0202(16) { I frame time out
,hdlc_p_timeout_cmd             = 0203(16) { P/F recovery attempt time out
,hdlc_e_timeout_cmd             = 0204(16) { Error recovery attempt time out
,hdlc_ia_timeout_cmd            = 0205(16) { Inactivity time out
,hdlc_ret_ex_cmd                = 0206(16) { Retransmit attempt count exceeded

          {=======================================}
          { ESCI itm command and response constants }
          {=======================================}

,esci_command_base              = 0300(16) { Command base for esci
,esci_startup_cmd               = 0301(16)
,esci_shutdown_cmd              = 0302(16)
,esci_suspend_cmd               = 0303(16)
,esci_resume_cmd                = 0304(16)
,esci_statistics_cmd            = 0305(16)
,esci_wakeup_cmd                = 0306(16)
,esci_switches_cmd              = 0307(16)
,esci_tdr_cmd                   = 0308(16)
,esci_diag_cmd                  = 0309(16)
,esci_nop_cmd                   = 030a(16)
,esci_dvmid_cmd                 = 030b(16)
,esci_dump_cmd                  = 030c(16)
,esci_xsub_cmd                  = 030d(16)

,esci_nures_res                 = 0320(16)
,esci_rcv_res                   = 0321(16)
,esci_xmit_res                  = 0322(16)
,esci_stistc_res                = 0323(16)
,esci_switches_res              = 0324(16)
,esci_tdr_res                   = 0325(16)
,esci_diag_res                  = 0326(16)
,esci_nop_res                   = 0327(16)
,esci_dump_res                  = 0328(16)
```

```
,esci_xsub_res                    = 0329(16)

        {=======================================}
        { SYSTEM ANCESTOR ITM WORKCODE DEFINITIONS }
        {=======================================}

,sa_start_task_for_user           = 0400(16) { Start a task on behalf of another task
,sa_reply                         = 0401(16) { Call start_system_task reply routine
,sa_dump_write                    = 0402(16) { Write data to dump file
,sa_dump_timer                    = 0403(16) { Time out dump processing
,sa_dump_close                    = 0404(16) { Close dump file
,sa_dump_restore                  = 0405(16) { Start dump processing and restore task
,sa_dump_only                     = 0406(16) { Start dump processing ( no restore )


        {=====================================}
        { SYSTEM AUDIT ITM WORKCODE DEFINITIONS }
        {=====================================}

,sys_audit_checksum               = 0450(16) { checksun system memory
,sys_audit_overflow               = 0451(16) { check user stack pointer for overflow
,sys_audit_report_the_mpb_status  = 0452(16) { check battery and temperature

{=================================================================}
{ MAINFRAME CHANNEL INTERFACE INTERTASK MESSAGE WORKCODE DEFINITIONS }
{=================================================================}

,mci_startup                      = 0501(16) { Specific MCI card
,mci_output_complete              = 0502(16) { PP has successful read
,mci_input_received               = 0503(16) { PP has successful write
,mci_data_available               = 0504(16) { Data is available for transfer
,mci_error_encountered            = 0505(16) { An error was found on a write
,mci_shutdown                     = 0506(16) { End processing
,mci_statistics                   = 0507(16) { Sender requests statistics
,mci_report_statistics            = 0508(16) { Announce statistics response
,mci_link_status_change           = 0509(16) { new link status
,mci_timer_expiration             = 050a(16) { Response timer has expired
,mci_log_message                  = 050b(16) { Message is to be logged
,mci_failure_detected             = 050c(16) { Failure detected
,mci_run_diagnostics              = 050d(16) { Run diagnostics
,mci_master_clear_check           = 050e(16) { Master clear threshold check
,mci_driver_shutdown_complete     = 050f(16) { Driver shutdown processing complete
,mci_end_of_queue                 = 0510(16) { Last message in queue


   {=========================================================}
   { INITIALIZATION M-E INTERTASK MESSAGE WORKCODE DEFINITIONS }
   {=========================================================}

,ime_pdu                          = 0601(16) { 3A indication parameters
,ime_transient_timer_expired      = 0602(16) { Transient task timer expired
,ime_init                         = 0603(16) { Transient task initialization
,ime_last_itm                     = 0604(16) { Transient task's last message
,ime_request_empty_itm_q          = 0605(16) { Request for  ,ime_last_itm
```

```
,ime_inactive_timer_expired      = 0606(16) { Main task timer  xpired

    {=======================================================}
    { XEROX TRANSPORT INTERTASK MESSAGE WORKCODE DEFINITIONS }
    {=======================================================}

,xt_transmit                     = 0700(16) { transmit delayed data
,xt_retransmit                   = 0701(16) { retransmit normal data
,xt_expedited_retransmit         = 0702(16) { retransmit expedited data
,xt_inactivity                   = 0703(16) { send a probe or kill the connection
,xt_cid_timer                    = 0704(16) { kill previously disconnected con'ctn
,xt_incoming_data_to_cep         = 0705(16) { process packet for a connection
,xt_incoming_data_to_sap         = 0706(16) { process packet for a sap
,xt_local_disconnect             = 0707(16) { kill connection due to local action
,xt_set_up_timer                 = 0708(16) { set up connection timer

    {=======================================================}
    { CDCNET STATISTICS MANAGER MESSAGE WORKCODE DEFINITIONS }
    {=======================================================}

,csm_issue_statistics_req        = 0800(16) { request statistics to be reported
,csm_process_timer_req           = 0801(16) { process statistics timer call

 {=================================================================}
 { OPERATOR SUPPORT APPLICATION INTERTASK MESSAGE WORKCODE DEFINITIONS }
 {=================================================================}

,osa_from_operator               = 850(16) { command indication from operator
,osa_from_transport              = 851(16) { indication from transport
,osa_from_internet               = 852(16) { indication from internet
,osa_terminate_osa               = 853(16) { kill osa
,osa_configure_osa               = 854(16) { initialize osa
,osa_cmd_response_time_expired   = 855(16) { command time limit expired
,osa_cmd_proc_cmd_indication     = 856(16) { cmd notice to osa cmd processor
,osa_broc_response_time_expired  = 857(16) { broadcast command time limit expired
,osa_alarm_data                  = 858(16) { alarm indication from Dep. Alarm ME
,osa_format_message               = 859(16)  {formatting workcode
,osa_termination_close           = 860(16) { connection broken close operators

 {=================================================================}
 { K DISPLAY SUPERVISOR INTERTASK MESSAGE WORKCODE DEFINITIONS      }
 {=================================================================}

,kdisp_initialization            = 888(16) { used to bring up k_display_supervisor
,kdisp_osa_disp_req              = 889(16) { used to send display requests to KDISP
,kdisp_osa_ack_brk_req           = 88A(16) { acknowledge break requests use this
,kdisp_bip_ind                   = 88B(16) { indication from BIP
,kdisp_svm_cc_ind                = 88C(16) { call confirm indication from SVM

 {=================================================================}
 {    LOG SUPPORT APPLICATION INTERTASK MESSAGE WORKCODE DEFINITIONS  }
 {=================================================================}

,lsa_log_request_workcode        = 900(16) { request for logging
```

```
,lsa_log_connect_retry           = 901(16) { retry because of  ransport connect failure
,lsa_alarm_connect_retry         = 902(16) { retry because of transport connect failur
,lsa_log_directory_indication    = 903(16) { logging directory indication
,lsa_alarm_directory_indication  = 904(16) { alarming directory indication
,lsa_log_transport_indication    = 905(16) { logging transport indication
,lsa_alarm_transport_indication  = 906(16) { alarming transport indication
,lsa_log_formatting_workcode     = 907(16)

 {============================================================}
 {    INDEPENDENT LOG M-E INTERTASK MESSAGE WORKCODE DEFINITIONS   }
 {============================================================}

,ilog_task_initialize            = 920(16) { Task initialization
,ilog_bip_indication             = 921(16) { Process a BIP indication
,ilog_transport_data_indication  = 922(16) { Process a Transport data indication

 {============================================================}
 { SSR and COMMAND PROCESSOR  INTERTASK MESSAGE WORKCODE DEFINITONS }
 {              ITM range = 980(16) to 1049(16)                }
 {============================================================}

       {*********************************************}
       {NOTE: 980(16) thru 987(16) should be eventually }
       {      deleted once SSRs and CPs are updated to  }
       {      using new and correct ITMs.               }
       {*********************************************}

,ssr_init_ok_workcode            = 980(16) { SSR initialization completed ok
,ssr_init_error_workcode         = 981(16) { SSR initialization error
,ssr_init_start_port_service_err = 982(16) { SSR start port service error
,ssr_init_queue_cim_command_err  = 983(16) { SSR queue cim command error
,ssr_shutdown_error_workcode     = 984(16) { SSR shutdown error
,ssr_shutdown_ok_workcode        = 985(16) { SSR shutdown ok workcode
,ssr_reset_timer_req_workcode    = 986(16) { SSR reset request workcode
,ssr_timeout_workcode            = 987(16) { SSR timed out workcode

,cp_ssr_enabled                  = 988(16) {HDLC SSR has been enabled,
                                           {but not active
,cp_ssr_active                   = 989(16) {SSR is active
,cp_ssr_start_failed             = 98A(16) {SSR is unable to start. See
                                           {response for reason.
,cp_ssr_processing_request       = 98B(16) {SSR is still processing request
,cp_ssr_timeout                  = 98C(16) {3A Command Processor has timed
                                           {out without hearing from the SSR
,cp_ssr_stopped                  = 98D(16) {SSR has shutdown successfully
,cp_ssr_stop_failed              = 98E(16) {SSR was unable to stop. See
                                           {response for reason.
,ssr_start_service               = 98F(16) {CP requesting SSR to start service
,ssr_stop_service                = 990(16) {CP requesting SSR to stop service

 {=========================================================}
 {  CONFIGURATION STATUS REPORTER ITM WORKCODE DEFINITIONS  }
 {=========================================================}
```

```
,csr_report_time                = 1050(16) { time to report   nfiguration status

 {=====================================================}
 {    CLOCK M-E INTERTASK MESSAGE WORKCODE DEFINITIONS   }
 {=====================================================}

,ck_sync_clock                  = 1060(16) { synchronize clock
,ck_sync_complete               = 1061(16) { synchronization attempt complete
,ck_stop_independent_clock      = 1062(16) { cancel the independent clock on this sy
,ck_disconnect_connection       = 1063(16) { disconnect this connection
,ck_start_clock                 = 1064(16) { start the independent clock on this sys
,ck_clock_started               = 1065(16) { independent clock started
,ck_clock_stopped               = 1066(16) { independent clock stopped

 {=========================================================}
 {    NETWORK PRODUCTS INTERTASK MESSAGE WORKCODE DEFINITIONS   }
 {              ITM range = 1080(16) to 1139(16)               }
 {=========================================================}

,cp_npi_active                  = 1080(16) { Initialization of NP has completed succ
,cp_npi_start_failed            = 1081(16) { NP Interface was unable to start
,cp_npi_timeout                 = 1082(16) { NP Command Processor has timed out w/out
                                           { from BIP.
,cp_npi_stopped                 = 1083(16) { Shutdown of NP has completed successfull
,cp_npi_stop_failed             = 1084(16) { NP Interface was unable to stop. See re
                                           { for reason.
,npi_start_service              = 1085(16) { CP requests NP to start
,npi_stop_service               = 1086(16) { CP requests NP to stop
,cp_npi_cancel_failed           = 1087(16) { NP Interface unable to cancel
,cp_npi_canceled                = 1088(16) { NP Interface able to cancel

 {=================================================================}
 { DIAGNOSTICS INTERTASK MESSAGE WORKCODE DEFINITIONS               }
 {=================================================================}

,dgm_start_cimo_exec            = 1100(16) { used to start CIM online
,dgm_cimo_dummy_itm             = 1101(16) { CIM online dummy message
,dgm_cimo_external_res          = 1102(16) { CIM online external test
,dgm_cimo_state_prg_err         = 1103(16) { CIM online state prog error
,dgm_start_esco_exec            = 1104(16) { used to start ESCI online

 {=========================================================}
 { C7 SVM INTERTASK MESSAGE WORKCODE DEFINITIONS (1110-1131)}
 {=========================================================}

,sv_reg                         = 1110(16) { Regulation indication
,sv_clear                       = 1111(16) { Clear indication
,sv_call_confirm                = 1112(16) { Call confirm indication
,sv_call                        = 1113(16) { Call indication
,sv_rej                         = 1114(16) { Reject indication
,sv_confirm_clear               = 1115(16) { Confirm clear indication
,sv_open                        = 1116(16) { Open indication
,sv_close                       = 1117(16) { Close indication
,sv_cancel                      = 1118(16) { NP interface request to cancel
```

```
,sv_init                          = 1119(16) { Initialization   quest
,sv_down                          = 111A(16) { Request from NAM
,sv_term                          = 111B(16) { Term request
,sv_call_t                        = 111C(16) { Call request from terminals
,sv_clear_t                       = 111D(16) { Clear request from terminals
,sv_shutdown                      = 111E(16) { Shutdown request
,sv_connection_down               = 111F(16) { Connection down request from BIP
,sv_terminal_characteristics      = 1120(16) { req. Terminal Char. be sent to NAM
,sv_application_accounting_stats = 1121(16) { req. appl. acctg. stats sent to NAM
,sv_terminal_accounting_stats     = 1122(16) { req. term. acctg. stats sent to NAM

 {=========================================================}
 { C7 BIP INTERTASK MESSAGE WORKCODE DEFINITIONS (1132-113F)}
 {=========================================================}

,bp_sv_shutdown_complete          = 1132(16) { shutdown complete
,bp_svm_req                       = 1133(16) { SVM request for NAM
,bp_initr                         = 1134(16) { INITR from SVM
,bp_term                          = 1135(16) { Terminate connection request from SVM
,bp_send_back                     = 1136(16) { Request BIP to send BACks

                {==============================}
                {  DEPENDENT FILE ACCESS M.E.  }
                {==============================}

,initial_request_workcode         = 1140(16) { file access initial request
,subsequent_request_workcode      = 1141(16) { file access subsequent request
,xport_interface_workcode         = 1142(16) { IFA PDU delivered from du from ifa
,dir_translation_workcode         = 1143(16) { Directory translation indication
```

I COMPARE

```
{ Procedure Name: i_compare
{
{ PURPOSE:
{   This function implements the interim version of the #COMPARE intrinsic.
{
{ Call Format:
{       (*callc inxcmp)
{       result := i_compare(string1, string2);

  FUNCTION [XDCL, #GATE] i_compare ( {
        s1: string ( * );
        s2: string ( * )): - 1 .. 1;
```

I COMPARE COLLATED

```
{ Procedure Name: i_compare_collated
{
{ PURPOSE:
{   This function implements the interim version of the #COMPARE_COLLATED
{   intrinsic.
{
{ Call Format:
{       (*callc inxcmpc)
{       result := i_compare_collated(string1,string2,table);

  FUNCTION [XDCL, #GATE] i_compare_collated ALIAS 'i_ccomp' ( {
        s1: string ( * );
        s2: string ( * );
        table: string (256)): - 1 .. 1;
```

I SCAN

```
{ Procedure Name: i_scan
{
{ PURPOSE:
{   This procedure implements the #SCAN intrinsic.
{
{ Call Format:
{       (*callc inxscan)
{       i_scan (select,string,index,found_char);

  PROCEDURE [XDCL, #GATE] i_scan ( {
{       select: ↑set OF char;
        select: ↑packed array [char] OF boolean;
        s: string ( * );
    VAR index: integer;
    VAR found_char: boolean);
```

I TRANSLATE

```
{ Procedure Name: i_translate
{
{ PURPOSE:
{   This procedure implements the #TRANSLATE intrinsic.
{
{ Call Format:
{       (*callc inxtran)
{       i_translate(table,source,destination);

  PROCEDURE [XDCL, #GATE] i_translate ( {
        table: string (256);
        source: string ( * );
    VAR destination: string ( * ));
```

LOAD ABS MODULE AND DELAY

```
{ PROCEDURE NAME: load_abs_module_and_delay
{
{ PURPOSE:
{   given a module name, return the information required to load the module
{
{ CALL FORMAT:
{   *callc dlxlamd
{   load_abs_module_and_delay (module_name, smm_address, load_address,
{                       transfer_address, byte_size,
{                       absolute_module_found, error_response);
{
{ DESCRIPTION:
{   Given a module name, a search is made to obtain information pertaining
{ to it.  If the name is not an absolute module, the parameter
{ absolute_module_found  is returned false.  If the module is not already
{ loaded, the On-Line Loader routine is called to do so.  Any error
{ message from the loader is returned in the parameter -  error_response.
{ Otherwise, upon return: smm_address will contain the starting
{ address of the module in SMM;  load_address will contain the
{ address where module loading should begin;  transfer_address will contain
{ the address at which module execution begins;  byte_size will contain
{ the size of the module in bytes;  and absolute_module_found will be true.
{ The module use count is incremented to prevent module deloading.
{
{ NOTE:  If the parameter absolute_module_found is returned FALSE, it is
{ the USER'S responsibility to release the buffer chain returned in
{ error_resonse.condition.
{

  PROCEDURE [XDCL] load_abs_module_and_delay
    ({
        module_name: pmt$program_name;
     VAR smm_address: ↑cell;
     VAR load_address: dlt$68000_address;
     VAR transfer_address: dlt$68000_address;
     VAR byte_size: dlt$section_length;
     VAR absolute_module_found: boolean;
     VAR error_response: clt$status);
```

LOAD ABS MODULE AND PROCEED

```
{ PROCEDURE NAME: load_abs_module_and_proceed
{
{ PURPOSE:
{   given a module name, return the information required to load the module
{
{ CALL FORMAT:
{   *callc dlxlamp
{   load_abs_module_and_proceed(module_name, reply_procedure, request_id);
{
{ DESCRIPTION:
{   Given a module name , a search is made to obtain information pertaining
{ to it.  If the name is not an absolute module, the parameter of the reply_
{ procedure: absolute_module_found  is returned false.  If the module is not
{ already loaded, the On-Line Loader routine is called to do so.  The calling
{ procedure is allowed to continue work during loading.
{ The following parameters are returned via the reply_procedure.  Any error
{ message from the loader is returned in the parameter -  error_response.
{ Otherwise, upon return: smm_address will contain the starting
{ address of the module in SMM;  load_address will contain the
{ address where module loading should begin;  transfer_address will contain
{ the address at which module execution begins;  byte_size will contain
{ the size of the module in bytes;  and absolute_module_found will be true.
{ The module use count is incremented to prevent module deloading.
{
{ NOTE:  If the parameter absolute_module_found is returned FALSE, it is
{ the USER'S responsibility to release the buffer chain returned in
{ error_resonse.condition.
{


  PROCEDURE [XDCL] load_abs_module_and_proceed
    ({
         module_name: pmt$program_name;
         reply_procedure: ↑procedure ({
                                        request_id: ↑cell;
                                        absolute_module_found: boolean;
                                        smm_address: ↑cell;
                                        load_address: dlt$68000_address;
                                        transfer_address: dlt$68000_address;
                                        byte_size: dlt$section_length;
                                        error_response: clt$status);
         request_id: ↑cell);
```

LOAD CMD PROCESSOR AND DELAY

```
{  PROCEDURE NAME:  load_cmd_processor_and_delay
{
{  PURPOSE:
{    Given a command processor name, load a module unless currently loaded.
{
{  CALL FORMAT:
{    *callc dlxlcpd
{    load_cmd_processor_and_delay (entry_point_name, entry_point_found,
{                                  entry_address, task_info, error_response,
{                                  module_ptr);
{
{  DESCRIPTION:
{    Search to see if the indicated module is currently loaded.
{  If not send an intertask message to the On-Line Loader to load
{  the module.  If the load fails, an error message is returned in the
{  parameter   error_response.  The module_use_count is incremented to
{  prevent module deloading.  The task attribute block is found and
{  validated (defaults are used on error).  Entry point information
{  is returned.
{
{  NOTE:  If the parameter entry_point_found is returned FALSE, it is
{  the USER'S responsibility to release the buffer chain returned in
{  error_resonse.condition.
{

  PROCEDURE [XDCL] load_cmd_processor_and_delay
    ({
        entry_point_name: pmt$program_name;
     VAR entry_point_found: boolean;
     VAR entry_address: ↑dlt$entry_description;
     VAR task_info: task_attributes;
     VAR error_response: clt$status;
     VAR module_ptr: dlt$load_id_ptr);
```

LOAD CMD PROCESSOR AND PROCEED

```
{ PROCEDURE NAME: load_cmd_processor_and_proceed
{
{ PURPOSE:
{   given a command processor name, return the information required to load the
{ module.
{
{ CALL FORMAT:
{   *callc dlxlcpp
{   load_cmd_processor_and_proceed(entry_point_name, reply_procedure,
{                                  request_id, module_ptr);
{
{ DESCRIPTION:
{   Given a command processor name, a search is made to obtain information
{ pertaining to it.  If the module is not already loaded, the On-Line Loader
{ routine is called to do so.  The calling procedure is allowed to continue
{ work during loading.  The following parameters are returned via the
{ reply_procedure.  Any error message from the loader is returned in the
{ parameter - error_response.  Otherwise, upon return:  task_info will
{ be the record containing the task attributes of stack_size and priority.
{ entry_address will contain the entry point address.
{
{  NOTE:  If the parameter entry_point_found is returned FALSE, it is
{  the USER'S responsibility to release the buffer chain returned in
{  error_resonse.condition.
{

  PROCEDURE [XDCL] load_cmd_processor_and_proceed
    ({
         entry_point_name: pmt$program_name;
         reply_procedure: ↑procedure ({
                                        request_id: ↑cell,
                                        entry_point_found: boolean,
                                        entry_address: ↑dlt$entry_description,
                                        task_info: task_attributes,
                                        error_response: clt$status,
                                        module_ptr: dlt$load_id_ptr);
         request_id: ↑cell);
```

LOAD ENTRY POINT AND DELAY

```
{  PROCEDURE NAME:  load_entry_point_and_delay
{
{  PURPOSE:
{    Given an entry point name, load a module unless currently loaded.
{
{  CALL FORMAT:
{    *callc dlxlepd
{    load_entry_point_and_delay (entry_point_name, entry_point_found,
{                                entry_address, task_info, error_response,
{                                module_ptr);
{
{  DESCRIPTION:
{    Search to see if the indicated module is currently loaded.
{  If not send an intertask message to the On-Line Loader to load
{  the module.  If the load fails, an error message is returned in the
{  parameter   error_response.  The module_use_count is incremented to
{  prevent module deloading.  The task attribute block is found and
{  validated (defaults are used on error).  Entry point information
{  is returned.
{
{  NOTE:  If the parameter entry_point_found is returned FALSE, it is
{  the USER'S responsibility to release the buffer chain returned in
{  error_resonse.condition.
{

  PROCEDURE [XDCL] load_entry_point_and_delay
    ({
        entry_point_name: pmt$program_name;
     VAR entry_point_found: boolean;
     VAR entry_address: ↑dlt$entry_description;
     VAR task_info: task_attributes;
     VAR error_response: clt$status;
     VAR module_ptr: dlt$load_id_ptr);
```

LOAD ENTRY POINT AND PROCEED

```
{ PROCEDURE NAME: load_entry_point_and_proceed
{
{ PURPOSE:
{   given an entry point name, return the information required to load the
{ module.
{
{ CALL FORMAT:
{   *callc dlxlepp
{   load_entry_point_and_proceed(entry_point_name, reply_procedure,
{                                request_id, module_ptr);
{
{ DESCRIPTION:
{   Given an entry_point_name, a search is made to obtain information
{ pertaining to it.  If the module is not already loaded, the On-Line Loader
{ routine is called to do so.  The calling procedure is allowed to continue
{ work during loading.  The following parameters are returned via the
{ reply_procedure.  Any error message from the loader is returned in the
{ parameter  -  error_response.  Otherwise, upon return:  task_info will
{ be the record containing the task attributes of stack_size and priority.
{ entry_address will contain the entry point address.
{
{  NOTE:  If the parameter entry_point_found is returned FALSE, it is
{  the USER'S responsibility to release the buffer chain returned in
{  error_resonse.condition.
{

  PROCEDURE [XDCL] load_entry_point_and_proceed
    ({
         entry_point_name: pmt$program_name;
         reply_procedure: ↑procedure ({
                                       request_id: ↑cell,
                                       entry_point_found: boolean,
                                       entry_address: ↑dlt$entry_description,
                                       task_info: task_attributes,
                                       error_response: clt$status,
                                       module_ptr: dlt$load_id_ptr);
         request_id: ↑cell);
```

LOCK SEMAPHORE

```
{ PROCEDURE NAME:  lock_semaphore
{
{ PURPOSE:
{   Signal Test-and-Set Semaphore.
{
{ CALL FORMAT:
{   (*callc CMXMTSK)
{   lock_semaphore(address, status);
{
{ DESCRIPTION:
{   The Test and Set instruction is executed on the
{   semaphore address after setting the bus lock.
{
{   This function is provided to permit multiple processor acquisition
{   of data structures in a controlled manner.
{
{   Semaphore  is a word value.  The Test and Set instruction sets the sign
{   bit and determines whether or not it was previously set in a single
{   cycle, excluding other processors until the entire job is complete.
{   The resource must be acquired in this manner, but may be released by
{   simpling storing a zero in the word.
{
{   This call  has the following effects:
{
{   NAME:            TRAP NUMBER:      EFFECTS:
{
{   lock_semaphore       0             the resource is acquired, or
{                                      a failure is returned.
{
```

```
PROCEDURE [INLINE] lock_semaphore ( {
      s1: ↑cell;
  VAR status: boolean);


  FUNCTION [XREF] call_fast_bg ( {
        index: integer;
        s1: ↑cell): 0 .. 32767;
  status := (call_fast_bg (33, s1) > 0);

PROCEND lock_semaphore;
```

LOG MESSAGE ENABLED

```
{ FUNCTION NAME: log_message_enabled
{
{ PURPOSE:
{   Determine  if  transmission  of  a specified log message is
{   enabled.
{
{ CALL FORMAT:
{   (*callc lsxlogr)
{   log_message_enabled (log_message_number, priority)
{
{ RETURNS:
{   TRUE  if  transmission  of log message is enabled AND there
{   is not memory and buffer congestion, else FALSE.
{
{  DESCRIPTION:
{   The   CDCNET  System  Log  Message  Vector  is  checked  to
{   determine  if  the  specified  log  message  is enabled for
{   transmission.
{
{   Parameter            Description
{   log_message_number   This   parameter  identifies  the  log
{                        message id number.
{
{   priority             This   parameter  identifies  the  log
{                        request priority. This parameter will
{                        be ignored in Release 1.0.   Possible
{                        values include log_critical, log_high,
{                        log_medium, and log_low.
{
{  GLOBAL DATA REFERENCED:
{   log_message_vector - List  of  all  log  and alarm messages
{                        whose  transmission  is  enabled.   It
{                        reflects  the  logical  "OR"  of  log
{                        messages  in  individual log and alarm
{                        groups.


  FUNCTION [XDCL] log_message_enabled ( {
        log_message_number: log_msg_id_type;
        priority: log_priority): boolean;
```

LOG REQUEST

```
{ PROCEDURE: log_request
{
{ PURPOSE: CDNA Logging interface.
{
{ Call Format:
{   (*callc lsxlogr)
{   log_request(message_id, message);
{
{ DESCRIPTION:
{   This procedure is the interface supplied by the Log Support
{   Application for CDNA users to access logging and alarm services.
{   The message_id and message provided by the user are sent to the
{   Log Support Application to be written to the log file or displayed
{   as an alarm.
{
{ PARAMETER DESCRIPTION:
{    message_id - A unique sixteen (16) bit integer which identifies
{          each log message in a CDNA system.
{
{    message - Address of the buffer containing the variable part of
{          the log message.  If no variable part exists, NIL is passed.
{          The log_message must be in management data unit format.
{
{ EXIT CONDITION:
{    The buf_ptr, message, is returned as NIL to the caller.
{
{ GLOBAL DATA REFERENCED:
{   lsa_task_id
{   system_data
{   sys_cnfg.buffer_state
{   sys_cnfg.memory_state


  PROCEDURE [XDCL] log_request ( {
        message_id: log_msg_id_type;
    VAR message: buf_ptr);
```

MAYBE TASK

```
{ PROCEDURE NAME:  maybe_task
{
{ PURPOSE:
{   Maybe Task.
{
{ CALL FORMAT:
{     (*callc CMXMTSK)
{    maybe_task (module_ptr, task_attributes, start_at, task);
{
{ DESCRIPTION:
{   A task is started at a procedure entry point.  The parameter
{   passed to it is the address of a recovery control block
{   chain, which chain is empty.
{
{   The module_ptr is put into the TCB for the task.
{
{   Tasks which start other tasks via this call become parent tasks;
{   the offspring is referred to as the child.  The executive will
{   send the parent messages with work codes in the range 0..15
{   regarding errant children.
{
{   The Executive, in this case, returns control whether or not the
{   task was started. If the task could not be started, the value of
{   parameter, TASK, is returned as NIL.
{
{   Refer to Executive ERS sections 4.19 and 3.5.2.
{

  PROCEDURE [INLINE] maybe_task ( {
        module_ptr: dlt$load_id_ptr;
        task_attr: task_attributes;
        lex_level_zero_xdcl: ↑procedure;
    VAR task: task_ptr);
```

MDU TO ASCII

```
{ Procedure Name:  mdu_to_ascii
{
{ Purpose:  convert management_data_unit syntax to ASCII
{
{ Description:
{ This routine converts a buffer with management data syntax to a buffer
{ containing an ASCII string.  No extra data is added.  I.e., no extraneous
{ -CR- or -LF- 's are added to the converted data.  If they are desired they
{ must already be in the buffer to be converted.  Note that data is appended
{ to the receiving buffer.  If there is none, set the buffer pointer to NIL
{ first.
{
{ The various field types are converted as follows:
{       binary string:  ASCII 0's and 1's
{       binary octet:  converted to hexadecimal ASCII digits
{       character octets:  none (already is ASCII)
{       binary integer:  converted to decimal ASCII digits
{       binary unsigned integer: converted to decimal ASCII digits
{       bcd:  converted to decimal ASCII digits
{       format:  converted to -LF- / -CR- sequence
{
{ Call Format:
{       (*callc mexm2a)
{       mdu_to_ascii (mdubuf, msgbuf);
{
{ Exit Conditions:
{ returns:  buffer with ASCII string.
{
{ NOTES:
{ This routine will yield and retry if unable to obtain memory or
{ buffers. This does not cause any problems currently since the two
{ callers of this routine can be preempted ( Terminal Support and the
{ Local Console Formatter ).
{

  PROCEDURE [XDCL, #GATE] mdu_to_ascii ( {
    VAR mdubuf, { data to be converted
        msgbuf: buf_ptr); { ASCII data is appended to this buffer
```

MEMORY OWNER IDENTIFICATION DEFINITIONS

```
{  TABLE NAME:  memory owner identification definitions
{
{  DECK NAME:  CMDMOWN
{
{

              {=======================================}
              { Memory owner identification definitions }
              {=======================================}

 TYPE
   memory_owner_type = 0 .. 3fff(16);

{
{ NOTE:  values 0 .. 20 (16) are reserved for use by the Executive
{


 CONST
   moe$initial_memory_allocation              =    1(16),
   moe$initial_data_buffer_alloc              =    2(16),
   moe$initial_desc_buffer_alloc              =    3(16),
   moe$free_memory                            =    4(16),
   moe$timer_entry                            =    5(16),
   moe$task_control_block                     =    6(16),
   moe$stack_area                             =    7(16),
   moe$user_qcb                               =    8(16),
   moe$user_queue_entries                     =    9(16),
   moe$intertask_messages                     =   0a(16),
   moe$exception_vector                       =   0b(16),
   moe$statistic_sampling_entry               =   0c(16),
   moe$bad_memory_extent                      =   0d(16),

   moe$member_of_log_msg_queue                =   21(16),
   moe$removed_from_log_msg_queue             =   22(16),

   { The following ranges are ids for buffers allocated for intelligent
   { peripherals. The actual id is computed by adding the ip card slot
   { to the base value for the particular range.
   { The ranges are broken down as follows:
   {
   {    30(16) - 37(16) = Buffers have been allocated for intelligent
   {                      peripheral by DVM. They (should) reside in
   {                      the working and reserve buffer pools in the
   {                      DVCB for the IP.
   {
   {    38(16) - 3f(16) = Buffers have been picked up by the IP from the
   {                      buffer pools in the DVCB. They are 'owned' by
   {                      the IP (only the IP has a pointer to them).
   {
   {    40(16) - 47(16) = Buffers have been placed in a status packet by
   {                      an IP, awaiting return to an MPB task.
```

```
{
{     48(16) - 4f(16) = Reserved for IP debugging.
{

moe$allocated_for_ip                        = 30(16),
moe$owned_by_ip                             = 38(16),
moe$returned_by_ip                          = 40(16),
moe$for_ip_debugging                        = 48(16),
```

MESSAGE DEQUEUE

```
{ PROCEDURE NAME: message_dequeue
{
{ PURPOSE:
{   Extract a message from a task-level message queue.
{
{ CALL FORMAT:
{   (*callc CMXPQUE)
{   message_dequeue(queue,message,time_interval);
{
{ DESCRIPTION:
{   This is a special high speed dequeuing routine specifically
{   for the use of protocol drivers and inter-layer interfaces
{   where data traffic is enqueued.
{
{ ALGORITHM:
{   A Queue Control Block (type qcb@) is supplied by the user.
{   The messages are unlinked together via the descriptor field
{   "next_message", which is there expressly for this purpose.
{   noprempt is called on entry, and the current value of the
{   system binary clock is inspected to determine the amount of
{   time (in milliseconds, accurate to 100 milliseconds) that
{   the message remained in the queue↑.  This time is used for
{   statistical support by layer 4 entities, and for 'Maximum
{   Packet CSMDELA' management by layer 2 entities.
{

  PROCEDURE [XDCL] message_dequeue ( {
        queue: qcb_ptr;
    VAR message: buf_ptr;
    VAR time_inv3Erval: integer);
```

MESSAGE ENQUEUE

```
{ PROCEDURE NAME: message_enqueue
{
{ PURPOSE:
{   Place a message in a task-level message queue.
{
{ CALL FORMAT:
{   (*callc CMXPQUE)
{   message_enqueue(queue,message);
{
{ DESCRIPTION:
{   This is a special high speed enqueuing routine specifically
{   for the use of protocol drivers and inter-layer interfaces
{   where data traffic is enqueued.
{
{ ALGORITHM:
{   A Queue Control Block (type qcb@) is supplied by the user.
{   The messages are linked together via the descriptor field
{   "next_message", which is there expressly for this purpose.
{   noprempt is called on entry, and the current value of the
{   system binary clock is copied into the descriptor's time
{   stamp field.
{

  PROCEDURE [XDCL] message_enqueue ( {
        queue: qcb_ptr;    ,
        msg: buf_ptr);
```

MODIFY WRITE PROTECT BYTE

```
{ PROCEDURE NAME: modify_write_protect_byte
{
{ PURPOSE: modify a byte in MPB write-protected RAM
{
{ CALL FORMAT:
{  (*callc cmxmwpb)
{  modify_write_protect_byte (↑byte,new_value_for_byte);
{
{ DESCRIPTION:
{ Noprempt is called and MPB RAM write-protect is cleared.  The field
{ is updated.  MPB RAM write-protect is set and okprempt is called.
{
{ NOTE - This routine will only work on bytes.
{

  PROCEDURE [XDCL] modify_write_protect_byte ( {
        ptr: ↑cell; { pointer to 'write-protect' value in user mode
          value: 0 .. 0ff(16)); { new value
```

MODIFY WRITE PROTECT LONG WORD

```
{ PROCEDURE NAME: modify_write_protect_long_word
{
{ PURPOSE: modify a long word in MPB write-protected RAM
{
{ CALL FORMAT:
{  (*callc cmxmwps)
{  modify_write_protect_long_word(↑long_word,new_value_for_long_word);
{
{ DESCRIPTION:
{ Noprempt is called and MPB RAM write-protect is cleared.  The field
{ is updated.  MPB RAM write-protect is set and okprempt is called.
{
{ NOTE - This routine will only work on long words.
{

  PROCEDURE [XDCL, #GATE] modify_write_protect_long_word ( {
        ptr: ↑cell; { pointer to 'write-protect' value in user mode
          value: integer); { new value
```

MODIFY WRITE PROTECT SHORT WORD

```
{ PROCEDURE NAME: modify_write_protect_short_word
{
{ PURPOSE: modify a short word in MPB write-protected RAM
{
{ CALL FORMAT:
{  (*callc cmxmwps)
{  modify_write_protect_short_word(↑short_word,new_value_for_short_word);
{
{ DESCRIPTION:
{ Noprempt is called and MPB RAM write-protect is cleared.  The field
{ is updated.  MPB RAM write-protect is set and okprempt is called.
{
{ NOTE - This routine will only work on short words.
{

  PROCEDURE [XDCL] modify_write_protect_short_word ( {
        ptr: ↑cell; { pointer to 'write-protect' value in user mode
          value: 0 .. 0ffff(16)); { new value
```

MPB RAM TEMPLATE

```
{ TABLE NAME: MPB RAM template
{
{ PURPOSE: describe well known MPB RAM addresses
{
{ CALL FORMAT:
{   (*callc sidram)
{
{ NOTE:
{   The variable mpb_ram_ptr is defined and initialized in this common
{   deck.  It can be used to access the fields defined in MPB_RAM.
{   In order to use mpb_ram_ptr, code must be compiled with CHKNIL turned off.
{
{ ********************************************************** }
{                        NOTICE                              }
{  This deck is interdependent with deck "BTGWRAM".          }
{  Any changes to this deck or "BTGWRAM" should result       }
{   in corresponding modifications.                          }
{                                                            }
{ ********************************************************** }

  VAR
    mpb_ram_ptr: [STATIC, READ] ↑mpb_ram := NIL; { ↑MPB_RAM from byte address 0

  TYPE
    mpb_ram = packed record { description of mpb ram starting from address 0
      vector: array [1 .. 256] of ↑cell, { vector space
      system_id: system_id_type, { unique identifier for this hardware box
      system_id_checksum: 0 .. 0ffff(16), { system_id checksum
      table_format_version: 0 .. 0ffff(16), { version of this RAM table format
      status: 0 .. 0ff(16), { MPB status register low 4 bits (if NMI occurs)
      mpb_ram_zeroed: 0 .. 0ff(16), { MPB RAM zeroed flag
      smm_size: integer, { # contiguous usable SMM bytes from 100,000(16)
      boot_map_entry_address: ↑cell, { ↑map entry used as bootstrap card
      auto_dump_table_address: ↑cell, { ↑ Auto Dump Table
      reset_status: 0 .. 0ff(16), { reset status saved from most current reset
      reset_code: 0 .. 0ff(16), { reset code ( from both software and hardware
     { see NOTE below
      software_error_code: 0 .. 0ff(16), { software error code
      hardware_reset_code: 0 .. 0ff(16), { possible hardware cause for reset
      version: 0 .. 0ffff(16), { version within last accepted help offer
      network_id: integer, { network id within last accepted help offer
      help_system_id: system_id_type, { system id within last accepted help
      auto_dump_subroutine_address: ↑cell, { ↑ Auto Dump Table generator
      auto_dump_subroutine_length: 0 .. 0ffff(16), { length in 16-bit words
      auto_dump_subroutine_checksum: 0 .. 0ffff(16), { 16-bit ones complement
      map_table: ALIGNED array [1 .. 72] of integer, { card map table
      reserved_4_bytes: integer, { reserved for future use
      mpb_error_routine_pointer: ↑cell, { starting address of MPB error routine
      mpb_error_routine_length: 0 .. 0ffff(16), { length in 16-bit words
      pmm_error_routine_pointer: ↑cell, { starting address of error routine
      pmm_error_routine_length: 0 .. 0ffff(16), { length in 16-bit words
      smm_error_routine_pointer: ↑cell, { starting address of error routine
```

```
    smm_error_routine_length: 0 .. 0ffff(16), { length in 16-b   words
    expected_smm_interrupt_flag: ↑cell, { expected SMM interrupt flag pointer
    ept_address: ALIGNED ↑cell, { starting address of the entry point table
    loaded_module_list: ↑dlt$module_header, { pointer to 1st entry
    unsatisfied_externals: ↑cell, { ↑ unsatisfied externals table
    desbuflen: integer, { length of descriptor buffers
    datbuflen: ALIGNED integer, { length of data buffers
    reserved_memory:ALIGNED 0 .. 32767, {reserved memory for critical use
    initial_loader_checksum: ALIGNED 0 .. 0ffff(16), {
    sys_cnfg_ptr: ↑cell, { address of executive configuration table
    system_ancestor_task_id: ALIGNED task_ptr, { ↑system ancestor tcb
    current_3b_ephemeral_sapid: ALIGNED sap_id_type, { next 3b dynamic SAP to assign
  recend;

{ The following is a kludge to allow addressing of software_error_code
  CONST
    software_error_address = 41a(16); { ↑mpb_ram_ptr↑.software_error_code
```

M_RELEASE

```
{ PROCEDURE NAME:  m_release
{
{ PURPOSE:
{   Multiple Release.
{
{ CALL FORMAT:
{   (*callc CMIPMLR)
{   m_release (buffer_address, c);
{
{ DESCRIPTION:
{   The equivalent of the specified number of message release
{   operations is performed.
{

  PROCEDURE [INLINE] m_release ( {
    VAR message: buf_ptr;
        c: 1 .. 32767);
```

NAME MATCH

```
{ FUNCTION NAME: name_match
{
{ PURPOSE:
{   Compare the two strings entered by checking for model conformity.
{
{ CALL FORMAT:
{   (*callc csxpnam)
{   result := name_match(name,model);
{
{ DESCRIPTION:
{   This function compares the two strings entered, name and model.
{   The name string may contain wild card attributes, the model string
{   is used to compare against the name string.  If the two strings
{   conform (match) the function returns a TRUE value; oherwise, it
{   returns FALSE.
{
{   Inputs:
{     name   string(*)   the name to be compared
{     model  string(*)   the model to compare against
{
{   The following characters have special meaning.  These characters
{   may be used in the name string as wild card entries.
{
{     [ ... ]  any single character among those in brackets
{
{     [↑... ]  any single character except those in brackets
{
{      a..z    within a bracketed group, a range of characters
{              is represented with two consecutive periods, i.e.:
{              "a..z", where "a" and "z" are any two characters for
{              which the expression a <= z or a >= z is accepted
{
{      *       any character string including the NULL string
{
{      ?       any single character
{
{      '       If the model contains any special characters,
{              those special characters (*, [, ?) must be surrounded
{              with single quotes.  If the model contains a single
{              quote, 2 single quotes must be in  the name.
{              example: the name string A'*'B  matches  the model
{              string A*B and the name string A''B matches the model
{              string A'B.
{
{   NOTE: If a '?' special character is followed by an '*' special
{         character (i.e: *?) the '*' special character is considered
{         the NULL string.
{
{         Special characters are not recgonized within a bracketed group.
{

  FUNCTION [XDCL] name_match ( {
```

```
name,
model: string ( * <= max_name_size)): boolean;
```

NEW INTERRUPT

```
{ PROCEDURE NAME:  new_interrupt
{
{ PURPOSE:
{   Announce Interrupt Service.
{
{ CALL FORMAT:
{   (*callc CMXMTSK)
{   new_interrupt (vector, server, task_id);
{
{ DESCRIPTION:
{   The interrupt vector table is modified to give control to
{   the interrupt service routine whenever the hardware interrupt
{   corresponding to the vector is invoked.
{
{   When this service is used, the task becomes a "parent task" to the
{   interrupt routine, which is referred to as the child.  The executive
{   will send the parent messages with work codes in the range 0..15
{   regarding errant children.
{

  PROCEDURE [INLINE] new_interrupt ( {
        vector: 2 .. 255;
        interupt_routine: ↑procedure;
    VAR task_owning_interrupt: task_ptr);



    FUNCTION [XREF] call_sure_bg ( {
          index: integer;
          vector: integer;
          interupt_routine: ↑procedure): task_ptr;
    task_owning_interrupt := call_sure_bg (75, vector, interupt_routine);
  PROCEND new_interrupt;
```

NEW PRIORITY

```
{ PROCEDURE NAME:  new_priority
{
{ PURPOSE:
{   Change Task Priority.
{
{ CALL FORMAT:
{   (*callc CMXMTSK)
{   new_priority (priority, task_id, status);
{
{ DESCRIPTION:
{   The task's priority is changed to the requested level.
{
{   Refer to Executive ERS section 4.23.
{

  PROCEDURE [INLINE] new_priority ( {
        requested_priority: priorities;
        task: task_ptr;
    VAR status: boolean);
```

NOPREMPT

```
{ PROCEDURE NAME:  noprempt
{
{ PURPOSE:
{   Suppress Task Preemption.
{
{ CALL FORMAT:
{   (*callc CMXPPRM)
{   noprempt:
{
{ DESCRIPTION:
{   A flag word in low memory is changed, signaling to the
{   executive TRAP 4 service routine that task preemption from
{   interrupt levels is disabled.
{
{ NOTES:
{   Refer to Executive ERS section 4.45.
{   Note that any executive call will restore task preemptability.

      PROCEDURE [XREF] noprempt;
```

OKPREMPT

```
{ PROCEDURE NAME:  okprempt
{
{ PURPOSE:
{   Restore Task Preemption.
{
{ CALL FORMAT:
{  (*callc CMXPPRM)
{  okprempt;
{
{ DESCRIPTION:
{   A flag word in low memory is changed, signaling to the
{   executive TRAP 4 service routine that task preemption from
{   interrupt levels is enabled.
{
{ NOTES:
{   Refer to Executive ERS section 4.46.
{   Note that any executive call will restore task preemptability.

      PROCEDURE [XREF] okprempt;
```

OPEN INTERNET SAP

```
{ PROCEDURE NAME:  open_internet_sap
{
{ PURPOSE:
{   Opens SAP entry for an INTERNET user.
{
{ CALL FORMAT:
{   (*callc b3xreqi)
{   open_internet_sap (input_param, output_param, return_code);
{
{ DESCRIPTION:
{   It is verified that Internet is up and the maximum number of SAPs are
{   already open.  If opening a dedicated SAP find_sap_entry is called
{   to verify that the SAP isn't already open.  If opening an ephemeral
{   SAP the next available SAP id is determined.  A SAP table entry is
{   created and internet SAP table built with the index to the new entry
{   inserted.
{
{ GLOBAL INPUT:
{   none
{
{ GLOBAL OUTPUT:
{   open_ephemeral_sap_count - number of ephemeral SAPs open
{   current_3b_ephemeral_sapid (in MPB_RAM) - next ephemeral SAP to assign
{   internet_sap_table - pointer to SAP table
{$

  PROCEDURE [#GATE, XDCL] open_internet_sap ( {
        input_param: ↑open_sap_input_parameters; { INPUT
        output_param: ↑open_sap_output_parameters; { INPUT
    VAR return_code: open_internet_sap_status); { OUTPUT - status of request
```

OPEN STATUS SAP

```
{ PROCEDURE:  open_status_sap
{
{ PURPOSE:
{   The  purpose  of  this procedure is to allow a software
{   component to register  the  address of its status table
{   by opening a software status sap.
{
{ CALL FORMAT:
{       (*callc sdxssar)
{       open_status_sap  (name,   task_id,   sap_table_ptr,
{       sap_number, status)
{
{ DESCRIPTION:
{   A     software     component     directly     calls     the
{   open_status_sap  routine after it is initialized and is
{   capable  of  reporting  status.  The  address  of  its
{   status tables is  placed in a status_sap_table upon the
{   open  which  can then  be  retrieved  by  the  software
{   components associated  command  processor  to  generate
{   the status of the software component.
{
{   Parameter Description
{   name: (input)
{       This  parameter is  the name of the software component.
{       The  name  provided  on the open_status_sap must be the
{       module name of the software  component if an associated
{       command processor is required by CDCNET.
{
{   task_id: (input)
{       This  parameter  identifies the task_id of the software
{       component who will open the software status sap.
{
{   sap_table_ptr: (input)
{       This  parameter  identifies the address of the software
{       components status table.
{
{   sap_number: (output)
{       This  is  a  return parameter which uniquely identifies
{       the status sap  opened.  The  sap_number  must be used
{       when later closing a status sap.
{
{   status: (output)
{       This  is  a return parameter which indicates if the sap
{       requested was  opened.  If  the sap was not opened try
{       again, but be warned that memory is low.
{
{ GLOBAL DATA REFERENCED:
{   software_status_sap_table
{
{ GLOBAL DATA MODIFIED:
{   software_status_sap_table
{
```

```
{ NOTES AND CAUTIONS:
{   The    procedure    NOPREMPT    is    called    upon    entering
{   open_status_sap      to      suppress      task      preemption.
{   Open_status_sap  is exited in a non-preemptable  state  and
{   will require the caller  to  make  a  call to the procedure
{   OKPREMPT if preemptability is so desired.


  PROCEDURE [XDCL] open_status_sap ( {
        name: string ( * <= 31);
        task_id: task_ptr;
        sap_table_ptr: ↑cell;
    VAR sap_number: software_sap_range;
    VAR status: access_status_type);
```

OPEN 3A SAP

```
{  PROCEDURE NAME:  open_3a_sap
{
{  PURPOSE:
{    This procedure is provided by Intranet to allow users to open an
{    Intranet SAP via a direct call.
{
{  CALL FORMAT:
{    (*callc a3xupl)
{    open_3a_sap (protocol_type, data_ind_proc, status_ind_proc,
{                 network_id, close_3a_sap_proc, data_request_3a_proc,
{                 sap, open_status);
{
{  DESCRIPTION:
{    A user of Intranet calls the open_3a_sap procedure directly.  The user
{    must provide its associated protocol type, the address of the procedure
{    Intranet calls to sent datagrams upline, the address of the procedure
{    Intranet calls to inform the user of changes in the (SSR) network
{    solution status, and the network_id the Intranet SAP is opened too
{    (NOTE: a network_id of zero indicates that the SAP is opened to all
{    network solutions).
{
{    A user of Intranet can elect not to receive any SSR status indications
{    by setting the status_ind_proc to NIL.
{
{    If the protocol type specified is out of range or its associated
{    SAP has already been opened then an error is returned to the Intranet
{    user via the open_status parameter and the error is logged.
{
{  RETURNS:
{    Name          Type                Description
{    close_3a_sap_proc                 Address of the 3A close_3a_sap subroutine.
{
{    data_request_3a_proc              Address of the 3A data_request_3a subroutine.
{
{    sap           intranet_sap_type   A unique sap is returned to each user.  This
{                                      sap must be specified whenever a datagram
{                                      is to be transmitted downline or the sap is
{                                      to be closed.
{
{    open_status 13a_status_type       This parameter indicates the status of the
{                                      open_3a_sap request.
{
{  GLOBAL DATA REFERENCED:
{    sap_table
{    sap_table_initialized
{
{  GLOBAL DATA MODIFIED:
{    sap_table

  PROCEDURE [XDCL, #GATE] open_3a_sap ( {
        protocol_type: protocol_range_type;
        data_ind_proc: user_datagram_proc_type;
```

```
    status_ind_proc: user_status_proc_type;
    network_id: network_id_type;
VAR close_3a_sap_proc: close_3a_sap_proc_type;
VAR data_request_3a_proc: data_request_3a_proc_type;
VAR sap: intranet_sap_type;
VAR open_status: 13a_status_type);
```

OSV LOWER TO UPPER

```
{ Table Name:  osv_lower_to_upper
{
{ Purpose:
{ Lower to upper case character translation table
{
{ Call Format:
{       (*callc osxtl2u)

  VAR
    osv_lower_to_upper ALIAS 'osvtl2u': [XDCL, READ, #GATE] string (256) :=
      CHR (00) CAT CHR (01) CAT CHR (02) CAT CHR (03) CAT CHR (04) CAT CHR
      (05) CAT CHR (06) CAT CHR (07) CAT CHR (08) CAT CHR (09) CAT CHR (10)
      CAT CHR (11) CAT CHR (12) CAT CHR (13) CAT CHR (14) CAT CHR (15) CAT CHR
      (16) CAT CHR (17) CAT CHR (18) CAT CHR (19) CAT CHR (20) CAT CHR (21)
      CAT CHR (22) CAT CHR (23) CAT CHR (24) CAT CHR (25) CAT CHR (26) CAT CHR
      (27) CAT CHR (28) CAT CHR (29) CAT CHR (30) CAT CHR (31) CAT ' !"#$%&'''
      CAT '()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑_@ABCDEFGHI'
      CAT 'JKLMNOPQRSTUVWXYZ{|}↑' CAT CHR (127) CAT CHR (128) CAT CHR (129)
      CAT CHR (130) CAT CHR (131) CAT CHR (132) CAT CHR (133) CAT CHR (134)
      CAT CHR (135) CAT CHR (136) CAT CHR (137) CAT CHR (138) CAT CHR (139)
      CAT CHR (140) CAT CHR (141) CAT CHR (142) CAT CHR (143) CAT CHR (144)
      CAT CHR (145) CAT CHR (146) CAT CHR (147) CAT CHR (148) CAT CHR (149)
      CAT CHR (150) CAT CHR (151) CAT CHR (152) CAT CHR (153) CAT CHR (154)
      CAT CHR (155) CAT CHR (156) CAT CHR (157) CAT CHR (158) CAT CHR (159)
      CAT CHR (160) CAT CHR (161) CAT CHR (162) CAT CHR (163) CAT CHR (164)
      CAT CHR (165) CAT CHR (166) CAT CHR (167) CAT CHR (168) CAT CHR (169)
      CAT CHR (170) CAT CHR (171) CAT CHR (172) CAT CHR (173) CAT CHR (174)
      CAT CHR (175) CAT CHR (176) CAT CHR (177) CAT CHR (178) CAT CHR (179)
      CAT CHR (180) CAT CHR (181) CAT CHR (182) CAT CHR (183) CAT CHR (184)
      CAT CHR (185) CAT CHR (186) CAT CHR (187) CAT CHR (188) CAT CHR (189)
      CAT CHR (190) CAT CHR (191) CAT CHR (192) CAT CHR (193) CAT CHR (194)
      CAT CHR (195) CAT CHR (196) CAT CHR (197) CAT CHR (198) CAT CHR (199)
      CAT CHR (200) CAT CHR (201) CAT CHR (202) CAT CHR (203) CAT CHR (204)
      CAT CHR (205) CAT CHR (206) CAT CHR (207) CAT CHR (208) CAT CHR (209)
      CAT CHR (210) CAT CHR (211) CAT CHR (212) CAT CHR (213) CAT CHR (214)
      CAT CHR (215) CAT CHR (216) CAT CHR (217) CAT CHR (218) CAT CHR (219)
      CAT CHR (220) CAT CHR (221) CAT CHR (222) CAT CHR (223) CAT CHR (224)
      CAT CHR (225) CAT CHR (226) CAT CHR (227) CAT CHR (228) CAT CHR (229)
      CAT CHR (230) CAT CHR (231) CAT CHR (232) CAT CHR (233) CAT CHR (234)
      CAT CHR (235) CAT CHR (236) CAT CHR (237) CAT CHR (238) CAT CHR (239)
      CAT CHR (240) CAT CHR (241) CAT CHR (242) CAT CHR (243) CAT CHR (244)
      CAT CHR (245) CAT CHR (246) CAT CHR (247) CAT CHR (248) CAT CHR (249)
      CAT CHR (250) CAT CHR (251) CAT CHR (252) CAT CHR (253) CAT CHR (254)
      CAT CHR (255);
```

OSV UPPER TO LOWER

```
{ Table Name:  osv_upper_to_lower
{
{ Purpose:
{ Upper to lower case character translation table.
{
{ Call Format:
{       (*callc osxtu21)

  VAR
    osv_upper_to_lower ALIAS 'osvtu21': [XDCL, READ, #GATE] string (256) :=
      CHR (00) CAT CHR (01) CAT CHR (02) CAT CHR (03) CAT CHR (04) CAT CHR
      (05) CAT CHR (06) CAT CHR (07) CAT CHR (08) CAT CHR (09) CAT CHR (10)
      CAT CHR (11) CAT CHR (12) CAT CHR (13) CAT CHR (14) CAT CHR (15) CAT CHR
      (16) CAT CHR (17) CAT CHR (18) CAT CHR (19) CAT CHR (20) CAT CHR (21)
      CAT CHR (22) CAT CHR (23) CAT CHR (24) CAT CHR (25) CAT CHR (26) CAT CHR
      (27) CAT CHR (28) CAT CHR (29) CAT CHR (30) CAT CHR (31) CAT ' !"#$%&'''
      CAT '()*+,-./0123456789:;<=>?@abcdefghijklmnopqrstuvwxyz[\]↑_@abcdefghi'
      CAT 'jklmnopqrstuvwxyz{|}↑' CAT CHR (127) CAT CHR (128) CAT CHR (129)
      CAT CHR (130) CAT CHR (131) CAT CHR (132) CAT CHR (133) CAT CHR (134)
      CAT CHR (135) CAT CHR (136) CAT CHR (137) CAT CHR (138) CAT CHR (139)
      CAT CHR (140) CAT CHR (141) CAT CHR (142) CAT CHR (143) CAT CHR (144)
      CAT CHR (145) CAT CHR (146) CAT CHR (147) CAT CHR (148) CAT CHR (149)
      CAT CHR (150) CAT CHR (151) CAT CHR (152) CAT CHR (153) CAT CHR (154)
      CAT CHR (155) CAT CHR (156) CAT CHR (157) CAT CHR (158) CAT CHR (159)
      CAT CHR (160) CAT CHR (161) CAT CHR (162) CAT CHR (163) CAT CHR (164)
      CAT CHR (165) CAT CHR (166) CAT CHR (167) CAT CHR (168) CAT CHR (169)
      CAT CHR (170) CAT CHR (171) CAT CHR (172) CAT CHR (173) CAT CHR (174)
      CAT CHR (175) CAT CHR (176) CAT CHR (177) CAT CHR (178) CAT CHR (179)
      CAT CHR (180) CAT CHR (181) CAT CHR (182) CAT CHR (183) CAT CHR (184)
      CAT CHR (185) CAT CHR (186) CAT CHR (187) CAT CHR (188) CAT CHR (189)
      CAT CHR (190) CAT CHR (191) CAT CHR (192) CAT CHR (193) CAT CHR (194)
      CAT CHR (195) CAT CHR (196) CAT CHR (197) CAT CHR (198) CAT CHR (199)
      CAT CHR (200) CAT CHR (201) CAT CHR (202) CAT CHR (203) CAT CHR (204)
      CAT CHR (205) CAT CHR (206) CAT CHR (207) CAT CHR (208) CAT CHR (209)
      CAT CHR (210) CAT CHR (211) CAT CHR (212) CAT CHR (213) CAT CHR (214)
      CAT CHR (215) CAT CHR (216) CAT CHR (217) CAT CHR (218) CAT CHR (219)
      CAT CHR (220) CAT CHR (221) CAT CHR (222) CAT CHR (223) CAT CHR (224)
      CAT CHR (225) CAT CHR (226) CAT CHR (227) CAT CHR (228) CAT CHR (229)
      CAT CHR (230) CAT CHR (231) CAT CHR (232) CAT CHR (233) CAT CHR (234)
      CAT CHR (235) CAT CHR (236) CAT CHR (237) CAT CHR (238) CAT CHR (239)
      CAT CHR (240) CAT CHR (241) CAT CHR (242) CAT CHR (243) CAT CHR (244)
      CAT CHR (245) CAT CHR (246) CAT CHR (247) CAT CHR (248) CAT CHR (249)
      CAT CHR (250) CAT CHR (251) CAT CHR (252) CAT CHR (253) CAT CHR (254)
      CAT CHR (255);
```

PCOPY

```
{ PROCEDURE NAME: pcopy
{
{ PURPOSE
{   Physical copy of Message To New Buffer Chain.
{
{ CALL FORMAT:
{   (*callc CMXPCPY)
{   pcopy (message, threshold);
{
{ DESCRIPTION:
{   The message is physically copied to new buffers, and the old
{   set of buffers is released.  Data is compact in the new
{   buffers; the first (n-1) buffers are full, and the last one
{   has all of its empty space in the trailing portion of the
{   buffer.
{
{   This is a highly time consuming operation, requiring at
{   least 3-5 microseconds per byte copied.  It is recommended
{   that the caller either run at a relatively low task
{   priority, or yield control sometime after the routine
{   returns to avoid time slice overrun, and to permit other
{   processes to be active.
{

  PROCEDURE [XDCL] pcopy (,{
    VAR message: buf_ptr;
        threshold: threshold_size;
    VAR success: boolean);
```

PICK

```
{ PROCEDURE NAME: pick
{
{ PURPOSE:
{   Remove a Structure from the Tree.
{
{ CALL FORMAT:
{   (*callc CMXPPIC)
{   addr := pick(head,key);
{
{ DESCRIPTION:
{   Locate a key in the tree, remove it from the tree, and
{   return the associated data entry, or NIL.
{

  PROCEDURE [XDCL] pick ( {
        head: ↑root; { root of tree
        key: integer) { key to be picked from tree
    ↑ cell; { table associated with key
```

PMP GET DATE

```
{ Procedure Name: pmp_get_date
{
{    The purpose of this request is to obtain the current date in
{  a user selected format.
{
{ Call Format:
{       (*callc pmxgdat)
{        PMP_GET_DATE (FORMAT, DATE, DATE_STR_LEN);
{
{  FORMAT: (input) This parameter specifies the format in which the date
{        will be returned.  Valid specifications are:
{          osc$month_date : month DD, YYYY
{            example: November 13, 1978
{          osc$mdy_date : MM/DD/YY
{            example: 11/13/78
{          osc$iso_date : YYYY-MM-DD
{            example: 1978-13-11
{          osc$ordinal_date : YYYYDDD
{            example: 1978317
{          osc$dmy_date : DD/MM/YY
{            example: 13/11/78
{          osc$default_date : an installation specified format from the above.
{{  DATE: (output) This parameter specifies the current date.
{
{  DATE_STR_LEN: (output) This parameter specifies the length of DATE.

  PROCEDURE [XDCL, #GATE] pmp_get_date ALIAS 'pmpgdat' ( {
        format: ost$date_formats;
    VAR date: ost$date;
    VAR date_str_len: 1 .. 18);
```

PMP GET TIME

```
{ Procedure Name:  pmp_get_time
{
{    The purpose of this request is to obtain the current time of day
{  in a user selected format.
{
{ Call Format:
{       (*callc pmxgtim)
{        PMP_GET_TIME (FORMAT, TIME, TIME_STR_LEN)
{
{  FORMAT: (input) This parameter specifies the format in which the time
{        will be returned.  Valid specifications are:
{          osc$ampm_time :  HH:MM AM or PM
{            example:  1:15 PM
{          osc$hms_time : HH:MM:SS
{            example: 13:15:21
{          osc$millisecond_time : HH:MM:SS:MMM
{            example: 13:15:21:453
{          osc$default_time : an installation specified format from the above.
{{  TIME: (output) This parameter specifies the current time.
{
{  TIME_STR_LEN: (output) This parameter specifies the length of TIME.
{

  PROCEDURE [XDCL, #GATE] pmp_get_time ALIAS 'pmpgtim' ( {
       format: ost$time_formats;
    VAR time_str: ost$time;
    VAR time_str_len: 1 .. 12);
```

POOL BUFFERS

```
{ PROCEDURE NAME:  pool_buffers
{
{ PURPOSE:
{   Add Buffers to Journal Pool.
{
{ CALL FORMAT:
{   (*callc CMXMP00)
{   pool_buffers (buffer_address);
{
{ DESCRIPTION:
{   The supplied chain of buffers is added to the journaling pool.
{   It is necessary to replenish the journal pool periodically if
{   the journal buffers are being returned to the journaling task
{   and the task desires to continue journaling.

  PROCEDURE [INLINE] pool_buffers (buffer_chain_allocated: buf_ptr);


    PROCEDURE [XREF] call_fast_bg ( {
          index: integer;
          buffer_chain_allocated: buf_ptr);
    call_fast_bg (4, buffer_chain_allocated);
  PROCEND pool_buffers;
```

PREFIX

```
{ PROCEDURE NAME:  prefix
{
{ PURPOSE:
{   Adds header to front of message.
{
{ CALL FORMAT:
{   (*callc CMXPPRE)
{   prefix(size_of_hdr_to_prefix,addr_of_header,ptr_msg,
{          threshold,allocation_type,success);
{
{ DESCRIPTION: (prefix Header to Message)
{   If there is empty space available in the first buffer, the
{   passed header is placed in this available space (if the buffer
{   is singly used).
{
{   If the buffer is not singly used or if the header did not fit
{   in the first buffer, then the number of data buffers required
{   to accomodate the header is obtained from the executive at the
{   specified threshold.  In this case, the new header will
{   start on an even byte.
{
{   The header is then copied into the buffer(s), and the
{   buffer(s) are attached at the front of the message. The header
{   is back filled to facilitate insertion of the next header
{   (to be prefixed) in the same buffer.  The Count in Message
{   field of the descriptor is maintained.  If a conditional
{   request is made(i.e. preference type = conditional) and that
{   buffer request is not satisfied, then a failure status is
{   returned.

  PROCEDURE [XDCL] prefix ( {
          size: non_empty_message_size; { #size(record to be stored)
          address: ↑cell; { ↑record to be stored
      VAR old: buf_ptr; { ↑existing message or NIL
          threshold: threshold_size, {buffer threshold to use
          allocation_type: pref_type; {absolute@,conditional
      VAR success: boolean); { type of return
```

PUT STATUS RECORD

```
{  PROCEDURE NAME: put_status_record
{
{  PURPOSE:
{    The purpose of this procedure is update the status record for the
{    device name specified.
{
{  CALL FORMAT:
{    (*callc sdxgpsr)
{    put_status_record (device_status_record, owner_key, status)
{
{  DESCRIPTION:
{    The device name in the device_status_record provided is search for
{    in the System Status Tables.  If the device name is found and the
{    owner key matches the configuration table address provided when the
{    device was requested the status record is updated and the status
{    return parameter is set to TRUE.  If the device name cannot be found
{    or the owner key is not correct FALSE is returned to the caller in
{    the status field.
{
{    Parameter              Description
{    device_status_record   The status record to be updated.  Just the
{                           status field will be updated via this routine.
{
{    owner_key              This parameter is provided by the caller.  The
{                           owners key is the address of the associated
{                           configuration table which was provided on the
{                           request_hardware_device call.
{
{    status                 A boolean value returned to caller.  If the
{                           status record was updated TRUE is returned;
{                           otherwise, FALSE is returned.
{
{  GLOBAL DATA MODIFIED:
{    major_card_status_table
{    lim_status_table
{    port_status_table(s)
{    smm_bank_status_table(s)
{    pmm_bank_status_table


  PROCEDURE [XDCL] put_status_record ( {
        device_status_record: component_status_type;
        owner_key: ↑cell;
    VAR status: boolean);
```

READ BCD CLOCK

```
{ PROCEDURE NAME:  read_bcd_clock
{
{ PURPOSE:
{   Read BCD Clock.
{
{ CALL FORMAT:
{   (*callc CMXMTIM)
{   read_bcd_clock (↑time);
{
{ DESCRIPTION:
{   The real time clock is read.
{   Refer to Executive ERS section 4.17.
{
{ SEE ALSO:
{   Set BCD Clock
{   Read Binary Clock

  PROCEDURE [INLINE] read_bcd_clock (the_time: ↑bcd_time);
```

READ CLOCK

```
{ PROCEDURE NAME:  read_clock
{
{ PURPOSE:
{   Read Binary Clock.
{
{ CALL FORMAT:
{   (*callc CMXMTIM)
{   read_clock (t);
{
{ DESCRIPTION:
{   The binary clock is read to millisecond accuracy and the
{   value is returned.
{   Refer to Executive ERS section 4.18.
{
{ SEE ALSO:
{   Set BCD Clock
{   Read BCD Clock

  PROCEDURE [INLINE] read_clock (VAR the_time: integer);
```

RELEASE MESSAGE

```
{ PROCEDURE NAME:  release_message
{
{ PURPOSE:
{   Release a Chain of Data Buffers.
{
{ CALL FORMAT:
{   (*callc CMXPRLB)
{   release_message (buffer_address);
{   fg_release_message (buffer_address);
{
{ DESCRIPTION:
{   The executive function Release Data Buffer Chain (4.4) is called,
{   with the following type:
{
{    NAME:                    TRAP NUMBER:    EFFECTS:
{
{    release_message               0          the buffer(s) are released
{
{    fg_release_message            2          interrupt routine use only;
{                                             the buffer(s) are released
```

```
   PROCEDURE [XDCL] release_message (VAR message: buf_ptr);
```

REQUEST DIAGNOSTIC ENTRY

```
{  PROCEDURE NAME: request_diagnostic_entry
{
{  PURPOSE:
{    The purpose of this procedure is to obtain the address of the
{    System Status Table entry for the device name specified.
{
{  CALL FORMAT:
{    (*callc dgxahwd)
{    request_diagnostic_entry (device_name, kind_of_status_table,
{         system_status_table_ptr, status);
{
{  DESCRIPTION:
{    The device name provided is parsed to determine its validity.
{    If a valid device name was specified and the associated board
{    type is physically available then the address of the associated
{    System Status Table is returned to the caller.
{
{    Parameter    Description
{    device_name  This parameter identifies the hardware device name
{                 being requested.
{
{    kind_of_status_table  This is a return parameter which
{                 identifies the type of System Status Table the
{                 system_status_table_ptr points too.
{
{    system_status_table_ptr  This is a return parameter which
{                 identifies the address of the System Status Table
{                 associated with the device name specified.
{
{    status       This is a return parameter which indicates if the
{                 address of the device name's status table was
{                 returned (TRUE); otherwise, it indicates that the
{                 device name specified is not available in the DI
{                 (FALSE).
{
{  GLOBAL DATA REFERENCED:
{    major_card_status_table
{    lim_status_table
{    port_status_table(s)
{    smm_bank_status_table(s)
{    pmm_bank_status_table


  PROCEDURE [XDCL] request_diagnostic_entry ( {
        device_name: string (maximum_device_name_size);
    VAR kind_of_status_table: system_status_table_type;
    VAR system_status_table_ptr: ↑cell;
    VAR status: boolean);
```

RESET CODES FOR THE DI

```
{  TABLE NAME:  Reset codes for the DI
{
{  COMMON DECK NAME:  SIDRC
{
{  CAUTION:
{    Some reset  codes are specified in  other decks for  some
{    68000 assembler routines.  These other decks are noted in
{    parentheses  on the comments  with the reset codes below.
{    Be sure to coordinate changes that concern those routines
{    with this common deck.

  CONST

{ Software reset code range
     minimum_software_reset_code     = 010(16)
    ,maximum_software_reset_code     = 0ff(16)

{ Hardware
    ,power_up_reset                  =   0     { MPB ROM (OBDGLBS)
    ,manual_reset                    =   2     { "
    ,halt_memory_fault               =   3     { "
    ,dead_man_time_out               =   4     { "

{ Software
    ,load_software_too_big           = 010(16) { Initialization Bootstrap
    ,improper_first_module           = 011(16) { "
    ,unsatisfied_external            = 012(16) { Initial Loader
    ,sysconfig_not_loaded            = 013(16) { "
    ,post_load_routines_not_found    = 014(16) { "
    ,reset_at_end_of_quiesce         = 015(16) { Initialization Bootstrap
    ,unrecognizable_object_text      = 016(16) { Initial Loader
    ,duplicate_entry_point           = 017(16) { "
    ,task_error_no_recovery_proc     = 018(16) { System Ancestor
    ,task_error_exceed_max_recovers  = 019(16) { "
    ,task_error_unrecoverable        = 01a(16) { "
    ,no_configuration_file_obtained  = 01b(16) { Configuration File Procurer
    ,configuration_file_read_error   = 01c(16) { "
    ,not_enough_memory_for_buffers   = 01d(16) { Loader
    ,identification_record_expected  = 01e(16) { "
    ,unexpected_idr_encountered      = 01f(16) { "
    ,premature_eof_on_file           = 020(16) { "
    ,absolute_length_too_large       = 021(16) { "
    ,invalid_object_text_version     = 022(16) { "
    ,invalid_module_kind             = 023(16) { "
    ,invalid_module_attribute        = 024(16) { "
    ,invalid_section_ordinal         = 025(16) { "
    ,duplicate_section               = 026(16) { "
    ,invalid_section_kind            = 027(16) { "
    ,invalid_allocation_alignment    = 028(16) { "
    ,invalid_offset                  = 029(16) { "
    ,storage_allocation_failed       = 02a(16) { "
    ,undefined_section               = 02b(16) { "
```

```
,reference_outside_of_section     = 02c(16) { "
,invalid_address_kind             = 02d(16) { "
,invalid_number_of_bytes_spanned  = 02e(16) { "
,transfer_;Sym_entry_pt_not_found = 02f(16) { "
,parameter_verification_error     = 030(16) { "
,loader_table_not_found           = 031(16) { "
,kill_system_with_dump            = 032(16) { Operator command:
,kill_system_without_dump         = 033(16) { KILL_SYSTEM | KILS
,stop_executive                   = 034(16) { Executive - S/W error (EXDEQUA)
,module_checksum_is_invalid       = 035(16) { System Audits
,software_dead_stop               = 036(16) { DEAD_STOP - S/W error
,smm_double_bit_error             = 037(16) { Executive - H/W error (EXDEQUA)
,ac_low_error                     = 038(16) { "                     (EXDEQUA)
,temperature_shutdown_error       = 039(16) { "                     (EXDEQUA)
,reset_from_debugger              = 03A(16) { hardwired in Debugger (DBMDBUG)
,overflowed_stack                 = 03B(16) { Exec. / System Audits (EXDEQUA)
,system_data_not_found            = 03C(16) { Loader
,boot_file_media_mismatch         = 03D(16) { boot startup code
```

RESET DI

```
{ PROCEDURE NAME:  reset_di
{
{ PURPOSE: Reset the DI
{
{ CALL FORMAT:
{   (*callc CMXRDI)
{   reset_di(software_reset_code);
{
{ DESCRIPTION:
{ The software error code is stored in RAM.  The DI debugger is called
{ to inform the human debugger of the reset.  The executive directive
{ to reset the DI is called.  This will force the DEADMAN Timer to expire.

  PROCEDURE [XDCL, #GATE] reset_di ( software_reset_code: integer);
```

RESET RECOVERY PROCEDURE

```
{ Procedure Name: reset_recovery_procedure
{
{ Purpose: pop recovery block off of task recovery stack
{
{ Call Format:
{  (*callc cmisisa)
{  reset_recovery_procedure(↑error_recovery_procedure);
{
{ Description:
{ The recovery block is popped off of the recovery stack of the calling
{ task.
{
{ See Also:
{ set_recovery_procedure
{ dump_write
{ dump_close
{

  PROCEDURE [INLINE] reset_recovery_procedure (procedure_address: ↑procedure);
    { caller's recovery routine
```

RESTORE TASK

```
{ PROCEDURE NAME:  restore_task
{
{ PURPOSE:
{   Restore Task.
{
{ DESCRIPTION:
{   The task is restored from the 'suspend' state, which was entered
{   either via Suspend or Abort Task.
{
{   Refer to Executive ERS section 4.28.
{

  PROCEDURE [INLINE] restore_task ( {
        t: task_ptr;
    VAR status: boolean);
```

SEND EXPRESS

```
{ PROCEDURE NAME:  send_express
{
{ PURPOSE:
{   Send Intertask Message to Express Queue.
{
{ CALL FORMAT:
{   (*callc CMXMTSK)
{   send_express (size, address, target, status);
{   wsend_express (size, address, target, status);
{   fg_to_express (size, address, target, status);
{
{ DESCRIPTION:
{   The addressed data structure is copied to a buffer, which is
{   enqueued to the target task's express message queue in FIFO
{   order.  If the task is waiting for a message on this queue,
{   the message is copied directly to the waiting task's data space.
{   The following calls have the following effects:
{
{   NAME:            TRAP NUMBER:     EFFECTS:
{
{   send_express         0            message is sent to target task.
{
{   wsend_express        1            message is sent to target task,
{                                     which may preempt the running task
{                                     if its priority is higher.
{
{   fg_to_express        2            interrupt routine use only;
{                                     message is sent to target task.
{                                     If TRAP 4 is used on exit, the
{                                     target task may preempt the
{                                     running task.
{
```

```
PROCEDURE [INLINE] send_express ( {
      size_of_message: 1 .. 32767;
      inter_task_message: ↑cell;
      target_task: task_ptr;
  VAR status: boolean);
```

SEND NORMAL,

```
{ PROCEDURE NAME:  send_normal,
{                  wsend_normal,
{                  fg_to_normal
{
{ PURPOSE:
{   Send Intertask Message to Normal Queue.
{
{ CALL FORMAT:
{   (*callc CMXMTSK)
{   send_normal (size, address, target, status);
{   wsend_normal (size, address, target, status);
{   fg_to_normal (size, address, target, status);
{
{ DESCRIPTION:
{   The addressed data structure is copied to a buffer, which is
{   enqueued to the target task's normal message queue in FIFO
{   order.  If the task is waiting for a message on this queue,
{   the message is copied directly to the waiting task's data space.
{   The following calls have the following effects:
{
{   NAME:            TRAP NUMBER:    EFFECTS:
{
{   send_normal          0           message is sent to target task.
{
{   wsend_normal         1           message is sent to target task,
{                                    which may preempt the running task
{                                    if its priority is higher.
{
{   fg_to_normal         2           interrupt routine use only;
{                                    message is sent to target task.
{                                    If TRAP 4 is used on exit, the
{                                    target task may preempt the
{                                    running task.
{
```

```
PROCEDURE [INLINE] send_normal ( {
      size_of_message: 1 .. 32767;
      inter_task_message: ↑cell;
      target_task: task_ptr;
  VAR status: boolean);
```

SET BCD CLOCK

```
{ PROCEDURE NAME:  set_bcd_clock
{
{ PURPOSE:
{   Set BCD Clock.
{
{ CALL FORMAT:
{   (*callc CMXMTIM)
{   set_bcd_clock (↑time);
{
{ DESCRIPTION:
{   The real time clock is set to the requested time, and the
{   binary clock value is changed to meet it.
{   Refer to Executive ERS section 4.16.
{
{ SEE ALSO:
{   Read BCD Clock
{   Read Binary Clock

  PROCEDURE [INLINE] set_bcd_clock (the_time: ↑bcd_time);
```

SET BUFFER CHAIN OWNER

```
{  PROCEDURE NAME:  set_buffer_chain_owner
{
{  PURPOSE:
{    Set an owner identification into the executive allocation field for
{  each descriptor with its associated data buffer in a buffer chain.
{
{  CALL FORMAT:
{    (*callc csxsbco)
{    set_buffer_chain_owner (buffer_address, owner_id);
{
{  DESCRIPTION:
{    The owner identification is placed into the allocator id
{  field of each descriptor and data buffer in the buffer chain.
{  This is provided so memory and buffer audit may be accomplished meaningfully.
{
{  See also:  set_memory_owner


PROCEDURE [XREF] set_buffer_chain_owner ( {
                 buffer_address: buf_ptr;
                 owner_id: memory_owner_type);
```

SET MEMORY OWNER

```
{  PROCEDURE NAME:  set_memory_owner
{
{  PURPOSE:
{    Set an owner identification into the memory header.
{
{  CALL FORMAT:
{    (*callc csxsmo)
{    set_memory_owner (memory_address, owner_id);
{
{  DESCRIPTION:
{    The owner identification is placed into the allocator id
{  field of the memory header.
{


PROCEDURE [XDCL] set_memory_owner ( {
                 memory_address: ↑cell;
                 owner_id: memory_owner_type);
```

SET RECOVERY PROCEDURE

```
{ Procedure Name: set_recovery_procedure
{
{ Purpose: push recovery block onto task recovery stack
{
{ Call Format:
{  (*callc cmisisa)
{  set_recovery_procedure(recovery_block,↑error_recovery_procedure);
{
{ Description:
{ The recovery block is pushed onto the recovery stack of the calling
{ task.  The pointer to the task's error recovery procedure is put
{ into the recovery block.
{
{ See Also:
{ reset_recovery_procedure
{ dump_write
{ dump_close
{

  PROCEDURE [INLINE] set_recovery_procedure ( {
          recovery_block: ↑sat$recovery_block; { empty recovery block
          procedure_address: ↑procedure); { caller's recovery routine
```

SET TEST LIGHTS

```
{  PROCEDURE NAME:  set_test_lights
{
{  PURPOSE:
{    The purpose of this procedure is to set the MPB test light state for
{    on-line diagnostics.
{
{  CALL FORMAT:
{    (*callc dgxsml)
{    set_test_lights(on_off, test_slot, error_code) ;
{
{  DESCRIPTION:
{    This common routine handles the setting of mpb test lights for
{    on-line diagnostics.
{
{    Parameter     Description
{    on_off        This parameter identifies the action to be taken.
{                  ON means that the test lights should be set to indicate
{                  that an on-line diagnostic is running.  OFF means that
{                  the test lights should be set to indicate that no on-line
{                  diagnostic is running or a diagnostic fault exists.
{
{    slot          Major card slot number.
{
{    error_code    Diagnostic error code.  Zero means no error.
{                  Parameter only has meaning if on_off = off.
```

SET WRITE PROTECT

```
{ PROCEDURE NAME:  set_write_protect
{
{ PURPOSE:
{   Set the write protect flag
{
{ CALL FORMAT:
{   (*callc cmiswp)
{   set_write_protect;
{
{ SEE ALSO:
{   clear_write_protect
{
{ NOTE:
{   The proper use of this routine is in conjunction with clear_write_protect
{   The order of use should be:
{     clear_write_protect;
{     <modify the normally write protected area of memory>
{     set_write_protect;
{


  PROCEDURE [INLINE] set_write_protect;
    ptr_control_commands↑.set_write_protect := 0;
  PROCEND set_write_protect;
```

SFIND

```
{ PROCEDURE NAME: sfind
{
{ PURPOSE:
{   Find Table in Tree Table Access Structure.
{
{ CALL FORMAT:
{   (*callc CMXPFIN)
{   addr := sfind(head, key);
{
{ DESCRIPTION:
{   The tree table access structure is searched for the provided key.
{   if it is found, the associated table is returned; otherwise
{   the return is NIL. The table is returned interlocked. (i.e., task
{   pre-emption from interrupt levels is disabled.)
{
{ SEE ALSO:
{   find_copy

  PROCEDURE [XDCL] sfind ( {
        head: ↑root, { head root of tree
        key: ↑string ( * )) {key for searching operations }
    ↑ cell; {table address of associated table
```

SFIND FIRST

```
{ PROCEDURE NAME: sfind_first
{
{ PURPOSE:
{   CSMFIND Table with Key Greater than a Given
{   Key and Return Interlocked.
{
{ CALL FORMAT:
{   (*callc CMXPFNF)
{   table = sfind_first(head, key, qual, param);
{
{ DESCRIPTION:
{   Locate the first entry in the tree having the stringally
{   greater key than that specified.
{   If qual <> NIL, call qual↑ (table, param, boolean_val).  and
{   return the first key having a non-zero return.  Return the
{   key in key, and return the associated table, interlocked.
{
{ SEE ALSO:
{   find_first
{   sfind_next
{   find_next


  PROCEDURE [XDCL] sfind_first ( {
          head: ↑root; { root of tree                    '
      VAR key: ↑string (*); {key associated with entry - returned
          qual: ↑procedure ( {user specified test function
                             ptr: ↑cell;
                             param_ptr: ↑cell;
                         VAR bool: boolean);
          param: ↑cell) {parameter to pass to qual
    ↑ cell; { table address of associated table
```

SFIND NEXT

```
{ PROCEDURE NAME: sfind_next
{
{ PURPOSE:
{   sfind Table with Key Greater than a Given
{   Key and Return Interlocked.
{
{ CALL FORMAT:
{   (*callc CMXPFNX)
{   table = sfind_next(head, key, qual, param);
{
{ DESCRIPTION:
{   Locate the first entry in the tree having the stringally
{   greater key than that specified.
{   If qual <> NIL, call (*qual) (table, param) and return the
{   first key having a non-zero return.  Return the key in key,
{   and return the associated table, interlocked.
{
{ SEE ALSO:
{   sfind_first
{   find_first
{   find_next
{

  PROCEDURE [XDCL] sfind_next ( {
          head: ↑root; { root of tree
      VAR key: ↑string (*); {key associated with entry - returned
          qual: ↑procedure ( {user specified test function
                              ptr: ↑cell;
                              param_ptr: ↑cell;
                         VAR bool: boolean);
          param: ↑cell) {parameter to pass to qual
    ↑ cell; { table address of associated table
```

SFIND WILD CARDS

```
{ PROCEDURE NAME: sfind_wild_cards
{
{
{ PURPOSE:
{   Locate wild card matches in B-tree.
{
{ CALL FORMAT:
{   (*callc csxpfwc)
{   sfind_wild_cards(ptr,key,process_match,para);
{
{ DESCRIPTION:
{   The B-tree is searched for a wild card match.  When a match
{   is found the user supplied procedure is involked to process the match.
{   The user supplied procedure has two parameters, a pointer to the first
{   associated table and a boolean value.  Searching for wild card matches
{   will continue until all elements in the B-tree have been exhausted or
{   the quit_processing parameter in the user supplied  procedure is returned
{   FALSE.

{   This routine processes each node by calling the user supplied procedure.
{   The routine terminates when all nodes have been processed or when the user
{   supplied routine returns a value of TRUE via the quit_processing parameter.
{   Nodes are processed in order.  Nodes are linked in a list until it
{   is there turn to be processed.
{
```

SGROW

```
{ PROCEDURE NAME: sgrow
{
{ PURPOSE:
{   Add New Table to Tree Table Access Structure.
{
{ CALL FORAMT:
{   (*callc CMXPGRO)
{    addr := sgrow(head, key, table, size)
{
{ DESCRIPTION:
{   The tree is searched for an existing association between the
{   provided key and a table structure.  If such a one exists,
{   the associated table is returned, and no update is performed.
{   Otherwise, such an association is created, and NIL is returned.
{   The table is returned interlocked. (i.e., task pre-emption
{   from interrupt levels is disabled.)
{

  PROCEDURE [XDCL] sgrow ( {
          head: ↑root; { root of the tree
          key: ↑string ( * ); { key for searching operations
          t: ↑cell; { table to be added to the tree
          size: integer)↑ cell;
```

SIGNAL1 / ACQUIRE1

```
{ PROCEDURE NAME:  signal1 / acquire1
{                  signal2 / acquire2
{                  signal3 / acquire3
{                  signal4 / acquire4
{
{ PURPOSE:
{   Signal Test-and-Set Semaphore.
{
{ CALL FORMAT:
{   (*callc CMXMTSK)
{   signal1 (address1, status);
{   acquire1 (address1, status);
{   etc., up to 4 addresses
{
{ DESCRIPTION:
{   The Test and Set instruction is executed sequentially on the
{   semaphoree addresses until either the list is completed or one
{   of the semaphorees is found to be set.  In the latter case,
{   deadlock is avoided by clearing the accepted semaphorees to zero
{   prior to returning.
{
{   This function is provided to permit multiple processor acquisition
{   of data structures in a controlled manner.
{
{   Semaphores are byte values.  The Test and Set instruction sets bit 7
{   and determines whether or not is was previously set in a single
{   cycle, excluding other processors until the entire job is complete.
{   Resources must be acquired in this manner, but may be released by
{   simpling storing a zero in the byte.  The executive clears the entire
{   byte when it releases the resources.
{
{   The following calls have the following effects:
{
{   NAME:            TRAP NUMBER:      EFFECTS:
{
{   acquire(n)            1            control returns when the resource
{                                      list is entirely acquired.
{
{   signal(n)             0            the resources are acquired, or
{                                      a failure is returned.
{
```

```
  PROCEDURE [INLINE] signal1 ( {
       s1: ↑cell;
    VAR status: boolean);
```

SPICK

```
{ PROCEDURE NAME: spick
{
{ PURPOSE:
{   Remove a Structure from the Tree.
{
{ CALL FORMAT:
{   (*callc CMXPPIC)
{   addr := spick(head,key);
{
{ DESCRIPTION:
{   Locate a key in the tree, remove it from the tree, and
{   return the associated data entry, or NIL.
{

  PROCEDURE [XDCL] spick ( {
        head: ↑root; { root of tree
        key_string: ↑string ( * )) { key to be pick from tree
    ↑ cell; { table associated with key
```

START DUMP

```
{ PROCEDURE NAME: start_dump
{
{ PURPOSE: start an online dump
{
{ CALL FORMAT:
{  (*callc(cmxsisa)
{  start_dump(override_dump_control,dump_started,dump_identifier)
{
{ DESCRIPTION:
{ A dump task is started to handle the online dump.  The dump control
{ block associated with the task is returned for future calls to
{ dump_write and dump_close.
{
{ NOTEs - The common subroutines wait and wake_up are used internally.
{
{ SEE ALSO:
{ dump_write
{ dump_close
{

  PROCEDURE [XDCL, #GATE] start_dump (override_dump_control: boolean; { user override
      VAR dump_started: boolean; { was the dump started ?
      VAR dump_identifier: ↑cell); { ↑ dump control block
```

START NAMED TASK AND DELAY

```
{  PROCEDURE NAME:  start_named_task_and_delay
{
{  PURPOSE:
{    Given an entry point name, start the appropriate task.
{
{  CALL FORMAT:
{    *callc dlxsntk
{    start_named_task_and_delay(entry_point_name, task_started,
{                               task_id, error_response);
{
{  DESCRIPTION:
{    The entry point name is looked up in the currently loaded
{  modules.  If the name is absent then the loader feature is
{  called to load the module.  If the load fails then an error
{  is returned.  The module use count is incremented to prevent
{  module deloading.  The task attribute block is found and
{  validated (defaults are used on error).  The System Ancestor
{  procedure start_system_task is called to start the task.  The
{  task id of the started task is returned.
{
{  NOTE:  If the parameter task_started is returned FALSE, it is
{  the USER'S responsibility to release the buffer chain returned in
{  error_resonse.condition.
{
```

```
PROCEDURE [XDCL] start_named_task_and_delay
  ({
       entry_point_name: pmt$program_name;
   VAR task_started: boolean;
   VAR task_id: task_ptr;
   VAR error_response: clt$status);
```

START NAMED TASK AND PROCEED

```
{  PROCEDURE NAME:  start_named_task_and_proceed
{
{  PURPOSE:
{    Given an entry point name, start the appropriate task.
{  The calling task is allowed to continue work during loading.
{
{  CALL FORMAT:
{    *callc dlxsntk
{    start_named_task_and_proceed(entry_point_name, reply_procedure,
{                                 request_id);
{
{  DESCRIPTION:
{    The entry point name is looked up in the currently loaded
{  modules.  If the name is absent then the loader feature is
{  called to load the module.  The calling task is allowed to
{  continue work during loading.  If the load fails then an error
{  is returned.  The module use count is incremented to prevent
{  module deloading.  The task attribute block is found and
{  validated (defaults are used on error).  The System Ancestor
{  procedure start_system_task is called to start the task.  The
{  task id of the started task is returned via the reply procedure.
{
{  NOTE:  If the parameter task_started is returned FALSE, it is
{  the USER'S responsibility to release the buffer chain returned in
{  error_resonse.condition.
{


  PROCEDURE [XDCL] start_named_task_and_proceed
    ({
         entry_point_name: pmt$program_name;
         reply_procedure: ↑procedure (    request_id: ↑cell,
                                          task_started: boolean,
                                          task_id: task_ptr,
                                          error_response: clt$status);
         request_id: ↑cell);
```

START SYSTEM TASK

```
{ PROCEDURE NAME: start_system_task
{
{ PURPOSE : start task for user with system ancestor as parent
{
{ CALL FORMAT:
{  (*callc cmxsisa)
{  start_system_task(transfer_address,priority,stack_size,reply_procedure,request_id);
{
{ DESCRIPTION:
{ The system ancestor starts up a task with the parameters transfer_address,
{ priority, and stack_size.  The reply procedure is called from the
{ system ancestor task to communicate with the task that called
{ start_system_task.
{
{ NOTE - The supplied reply procedure should have minimal functionality
{        since it executes under the system ancestor task.
{

  PROCEDURE [XDCL, #GATE] start_system_task (transfer_address: ↑procedure, { task entry p
          task_attr: ↑task_attributes,
          reply_procedure: ↑procedure (request_id: ↑cell, task_id: task_ptr),
          request_id: ↑cell, { user request identifier to link request and response
          module_ptr: dlt$load_id_ptr);
```

START TASK

```
{ PROCEDURE NAME:  start_task
{
{ PURPOSE:
{   Start Task.
{
{ CALL FORMAT:
{     (*callc CMXMTSK)
{    start_task (module_ptr, task_attributes, start_at, task);
{
{ DESCRIPTION:
{   A task is started at a procedure entry point.  The parameter
{   passed to it is the address of a recovery control block
{   chain, which chain is empty.
{
{   The module_ptr is put into the TCB for the task.
{
{   Tasks which start other tasks via this call become parent tasks;
{   the offspring is referred to as the child.  The executive will
{   send the parent messages with work codes in the range 0..15
{   regarding errant children.
{
{   Refer to Executive ERS sections 4.19 and 3.5.2.
{

  PROCEDURE [INLINE] start_task ( {
        module_ptr: dlt$load_id_ptr;
        task_attr: task_attributes;
        lex_level_zero_xdcl: ↑procedure;
    VAR task: task_ptr);
```

STOP TASK

```
{ PROCEDURE NAME:  stop_task
{
{ PURPOSE:
{   Stop Task.
{
{ CALL FORMAT:
{   (*callc CMXMTSK)
{   stop_task (task, status);
{
{ DESCRIPTION:
{   The task is permanently removed from the system.
{
{   Refer to Executive ERS section 4.21.
{

  PROCEDURE [INLINE] stop_task ( {
        task: task_ptr;
    VAR status: boolean);
```

STRIP

```
{ PROCEDURE NAME: strip
{
{ PURPOSE:
{   Remove header from front of message.
{
{ CALL FORMAT:
{   (*callc CMXPSTR)
{   strip(hdr_size,addr_of_user_space,strip_msg_addr,
{         threshold);
{
{ DESCRIPTION:
{
{   The message is checked for use by multiple data streams.  If
{   the leading portion is so used, that portion is logically
{   copied.  The entire message is not logically copied unless
{   this is absolutely necessary.
{
{   The header is then copied into the text area, and any
{   emptied buffers are released.  The Count in Message field of
{   the descriptor is maintained.
{
{   In the event that a copy operation is required, the copied
{   buffers will be released before returning to the caller.
{
{   Strip differs from strip_in_place in that the passed user
{   space is always used and no attempt is made to not move the
{   stripped header space(i.e.  data is always moved).
{   Strip_in_place calls strip if data movement is required.
{
{

  PROCEDURE [XDCL] strip ( {
        hdr_size: non_empty_message_size;
        addr_of_user_space: ↑cell;
    VAR msg: buf_ptr;
        threshold: threshold_size);
```

STRIP IN PLACE

```
{ PROCEDURE NAME: strip_in_place
{
{ PURPOSE:
{   Return Header Address (without moving it if possible) and
{   "remove" header from message.
{
{ CALL FORMAT:
{   (*callc CMXPSIP)
{   strip_in_place(hdr_size,ptr_record,ptr_returned_record,
{                  strip_msg_addr,threshold);
{
{ DESCRIPTION:
{   If the header is contained in one buffer, begins on an even
{   byte boundry and is not multiply used, the header address is
{   returned and offset changed to remove the header.
{   Otherwise, strip is called to move the header to the users
{   area.
{

  PROCEDURE [XDCL] strip_in_place ( {
        isize: non_empty_message_size;
        address: ↑cell;
    VAR table: ↑cell;
    VAR msg: buf_ptr;
        threshold: threshold_size);
```

SUBFIELD

```
{ PROCEDURE NAME: Subfield
{
{ PURPOSE:
{   Obtain Multiple Byte Header Field(s) from Message.
{
{ CALL FORMAT:
{     (*callc CMXPSUB)
{     subfield (displacement, length, text, message);
{
{ DESCRIPTION:
{   The subfield is copied into the text area.
{
{ NOTES:
{   The parameter "message" may not be null (0). It must be
{   a valid descriptor buffer address.
{
{ SEE ALSO:
{   Trim, Prefix, Logical Copy, Strip, Bsubfield
{

  PROCEDURE [XDCL] subfield ( {
        displacement: message_size;
    VAR length: non_empty_message_size; {sizeof(text) number of bytes in
                                        {subfield
        text: ↑cell; {address of space subfield is copied to
        message: buf_ptr); {address of message to copy from
```

SUSPEND

```
{ PROCEDURE NAME:  suspend
{
{ PURPOSE:
{   Suspend Task.
{
{ CALL FORMAT:
{   (*callc CMXMTSK)
{   suspend (task, status);
{
{ DESCRIPTION:
{   The task is forced into an undispatchable state, but the parent
{   task is not notified.  This is similar to the normal function of
{   Abort Task, and is restored with the same call, but is intended
{   for use by another task which wishes to take matters into its
{   own hands.  It could also be used as an alternate form of wait/
{   wakeup, in the event that (for example) a directly called file
{   processor or similar program wished to make its caller wait for
{   completion without interdicting the normal intertask message and
{   wait/wakeup mechanism.
{
{   Refer to Executive ERS section 4.27.
{

  PROCEDURE [INLINE] suspend ( {
        t: task_ptr;
    VAR status: boolean);
```

SYSTEM CONFIGURATION TABLE

```
{ TABLE NAME: system configuration table
{
{ PURPOSE:
{   Describes System Configuration Parameters.
{
{ CALL FORMAT:
{   (*callc CMCCNFG)
{   VAR
{     address: ↑exec_config,
{     table: exec_config;
{

  TYPE
    exec_config = record
      maxprior: 0 .. 32767, { highest valid priority -- lowest is zero
      databac: 0 .. 32767, { data buffer available count
      descbac: 0 .. 32767, { descriptor buffer available count
      lbufflen: integer, { data space length in bytes
      sbufflen: integer, { descriptor buffer length in bytes
      stdstack: integer, { standard stack allocation
      running: task_ptr, { task_ptr of running task
      curprior: priorities, { currently running priority
      schprior: priorities, { highest scheduled priority
      pmtok: boolean, { task preemption permission flag
      vecslice: integer, { interrupt vector for time slice interrupt
      vecintvl: integer, { interrupt vector for interval timer interrupt
      vecclock: integer, { interrupt vector for millisecond interrupt
      mpbramtop: integer, {numerically largest address in mpb ram
      privatetop: integer, { numerically largest address in private memory
      globfree: integer, { number of bytes of free global memory
      locfree: integer, { number of bytes of free private memory
      mpbfree: integer, { number of bytes of free mpb ram memory
      globfrag: 0 .. 32767, { number of extents of free global memory
      locfrag: 0 .. 32767, { number of extents of free private memory
      mpbfrag: 0 .. 32767, { number of extents of free mpb ram memory
      deloadtyp: deload_flag, { type of memory to release flag for deload
                              { task
      deloadtcb: task_ptr, { task_ptr of deload task
      deloadmpb: 0 .. 0ffff(16), { deloadable bytes of mpb ram
      deloadpmm: integer, { deloadable bytes of private memory
      deloadsmm: integer, { deloadable bytes of global memory
      mpbthresh: 0 .. 0ffff(16), { dload threshold for mpb am
      pmmthresh: integer, { deload threshold for private memory
      smmthresh: integer, { deload threshold for global memory
      pmtreq: boolean, { task will yield on next trap 1 or trap 4 if set
      retryflag: 0 .. 32767, { retry in progress flag
      clocktyp: 0 .. 1, { 0 = millisecond clock; 1 = real time clock
      timertcb: task_ptr, { task_ptr of time task
      diagflag: 0 ..0ffff(16), { flags set in Traps to indicate call type
      binclock: integer, { .1 second accuracy binary time of day
      decclock: bcd_time, { .1 second accuracy bcd date/time
      assumed_year: 0 .. 32767, {assumed year used by executive
```

```
    firewall: integer, { address of interrupt firewall chain
    prilist: array [priorities] of qcb@, { ready lists for tasks scheduled at
                                          {priorities
    globmem: qcb@, { global memory extent list
    privmem: qcb@, { private memory extent list
    mpbmem: qcb@, { mpb ram memory extent list
    iptlist: qcb@, { defined interrupts list
    lbuffq: qcb@, { data buffer queue
    sbuffq: qcb@, { descriptor buffer queue
    data_buffer_count: 0 .. 32767, { number of data buffers
    descriptor_buffer_count: 0 .. 32767, { number of descriptor buffers
    expire_stp: 0 .. 32767, { expire state transition processor timer
    stack_overflow_space: integer, { size of stack overflow area allocated
    task_overflowed: task_ptr, { task_ptr of task which has overflowed its stack
    pc_chkinst_address: integer,  { PC where chk instruction executed
    usp_chkinst_address: integer, { USP when chk instruction executed
    mpb_light_state: light_status, { status of mpb lights
    idle_loop_count: integer, { executions of idle loop since last clear
    reservetop: integer, { numerically largest address in reserve memory
    rsvfree: integer, { number of bytes of reserve ram memory
    rsvfrag: 0 .. 32767, { number of extents of reserve global memory
    rsvmem: qcb@, { reserve ram memory extent list
    memory_state: memory_state_type, {depends on amount of free memory
    buffer_state: buffer_state_type, {depends on amount of free memory
    stp_timer: ↑timer,  {timer id of state transition processor
    cio_b: cio_port_b, {cio port b bit settings
    cio_c: ALIGNED cio_port_c, {cio port c bit settings
    supervisor_state_ok: 0 .. 0ffff(16), { 1 = ok, 0 = user task
  recend;

TYPE
  priorities = 0 .. max_priority,
  stack_size = min_stack_size .. max_stack_size;

CONST
  max_priority = 7,
  min_stack_size = 0,
  max_stack_size = 02000(16);

TYPE
  deload_flag = ( dlc$mpb, dlc$pmm, dlc$smm );
```

TASK CONTROL BLOCK

```
{ TABLE NAME:  task_control_block
{
{ PURPOSE:
{   Task Constants and Types.
{
{ CALL FORMAT:
{   (*callc CMDTTSK)
{   VAR
{     name: task_ptr;
{
{ DESCRIPTION:
{   This structure describes a task to the Executive.
{

  TYPE
    qcb@ = record
      length: 0 .. 32767, { current length of queue
      count: 0 .. 32767, { number of enqueues that have happened to this QCB
      qnext: buf_ptr,
      qlast: buf_ptr,
      qcharacters: integer, { number of characters in queue
    recend;

  TYPE
    qcb_ptr = ↑qcb@,
    qcb = ↑qcb@; { archaic; for C compatibility only

  TYPE
    taskid@ = packed record { packed to force correct data mappings
      next_task: task_ptr, { chain to next task_ptr
      id: integer, { = '!TCB'
      stsiz: integer, { size of current stack segment
      chldq: task_ptr, { task_ptr of my next sibling
      adult: task_ptr, { task_ptr of my parent
      child: task_ptr, { task_ptr of my child
      stack: integer, { address of my current stack segment
      state_fill: 0 .. 31,
      state: 0 .. 7, { my.current state
      transition_fill: 0 .. 15,
      trans: 0 .. 15, { transition that entered this state
      tran: array [0 .. 15] of 0 .. 65535, { counts of transitions to date
      slices: 0 .. 65535, { count of time slice overruns to date
      flag_fill_1: 0 .. 31,
      preempted: boolean, { task has been preempted; registers all saved (else
                            {only A6 and D7)
      hold: boolean, { used by timer task to deflect timer requests into
                     {"normal" queue
      wku: boolean, { wakeup pending if set
      flag_fill_2: 0 .. 255,
      express: qcb@, { inter-task message queue
      normal: qcb@, { inter-task message queue
      preempt_permit : 0 .. 32768, { zero = task not preemptable. any other
```

```
                                    {value = task preemptable
    cpriority: 0 .. 32768, { my nominal priority
    priority: 0 .. 32768, { my actual priority
    d_registers: array [0 .. 7] of integer, { only register D7 normally valid
      a_registers: array [0 .. 6] of ↑cell, { only register A6 normally valid
      usp: ↑cell, { user stack pointer
    sr: 0 .. 0ffff(16), { status register
    pc: ↑cell, { program counter
    tcbfrb: ↑sat$recovery_block, { pointer to task failure recovery block
    tcb_epa:  ↑cell, { task entry point address
    tcb_space: integer, { amount of unused space in reserved stack area
    tcbmhp: dlt$module_header_ptr, { pointer to module header
    age: 0 .. 0ffff(16), { age within dispatch queue
  recend;

TYPE
  taskid = ↑taskid@, { archaic; for C compatibility only
  task_ptr = ↑taskid@;
```

THRESHOLDS

```
{ TABLE NAME : thresholds
{
{ PURPOSE:
{  Buffer thresholds.
{
{ CALL FORMAT:
{  (*callc CMDTHRH)
{

  TYPE
    threshold_size = 0 .. 7;

  CONST
    svm_thresh= 0, { SVM buffer threshold
    rcv_threshold = 0; { buffer and descriptor thresholds used by whomsoever
```

TIMER ENTRIES

```
{ PROCEDURE NAME:  timer entries
{
{ PURPOSE:
{   Defines Timer Entries.
{
{ CALL FORMAT:
{   (*callc CMDTTIM)
{   VAR
{     t: ↑timer,
{     bcdtime: bcd_time;
{
{ DESCRIPTION:
{   This is the format of timer entries,intertask messages,
{   and qcbid elements.
{
{ CAUTION:  Any changes to the timer record must be reflected
{   in common deck EXDEQUA.

  TYPE
    timer = record
      next_one: ↑timer, { next timer in queue
      length: 0 .. 32767, { length of what follows
      mark: integer, { = '!TIM'
      code: 0 .. 15, { identifying code
      tod: milliseconds, { time of day to pop
      period: milliseconds, { period,if periodic timer
      param: ↑cell, { parameter for subroutine
      proc: ↑procedure, { address of subroutine
    recend;

  TYPE
    milliseconds = integer,
    timer_ptr = ↑timer,
    timerid = ↑timer; { archaic; for C compatibility only

  TYPE
    bcd = 0 .. 9,
    bcd_time = packed record
      lyear: bcd,
      ryear: bcd,
      lmonth: bcd,
      rmonth: bcd,
      lday: bcd,
      rday: bcd,
      lhour: bcd,
      rhour: bcd,
      lminute: bcd,
      rminute: bcd,
      lsecond: bcd,
      rsecond: bcd,
      deci: bcd,
      centi: bcd,
```

```
  milli: bcd,
recend;
```

TIME

```
{ PROCEDURE NAME:  time
{
{ PURPOSE:
{   Convert time_of_day to milliseconds.
{
{ CALL FORMAT:
{   (*callc CMXMTIM)
{   milliseconds := time (hours, minutes, seconds);
{
{ DESCRIPTION:
{   This function permits time_of_day and interval to be
{   specified in a readable manner.
{
{   E.g., midnight is either time (0, 0, 0) or time (24, 0 , 0).
{   1:53:22 PM is time (13, 53, 22),
{   an interval of 10 seconds is time (0, 0, 10)
{

  PROCEDURE [INLINE] time ( {
        hour: 0 .. 24,
        minute,
        second: 0 .. 59) milliseconds;
```

TRANSLATE MESSAGE

```
{ PROCEDURE NAME:   Translate_Message
{
{ PURPOSE:
{   Translate Message Character Set.
{
{ CALL FORMAT:
{   (*callc CMXPTRA)
{   translate_message (message, table, threshold);
{
{ DESCRIPTION:
{   The intended use of this routine is character set
{   translation, such as EBCDIC to ASCII, ASCII to Baudot, etc.
{   The translation table provides a mapping of the 'from'
{   character set to the 'to' character set.
{
{   The message is checked for multiple use.  If any portion is
{   multiply used, a set of buffers is obtained, and translation
{   is performed into the new set of buffers; otherwise,
{   translation is done in place.
{
{   In either case, translation is in effect the repeated
{   execution of the statement:
{      *to++ := table[*from++];
{   and is performed on a character-by-character basis.
{
{   In the event that a copy operation is required, the copied
{   buffers will be released before returning to the caller.
{
{ NOTES:
{   The addresses for "message" and "table" must be valid. "table"
{   will normally be a read-only static data structure.
{
{   This is a highly time consuming operation, requiring a minimum of
{   5 microseconds per character translated. It is recommended that
{   the caller yield control sometime after returning to avoid time
{   slice overrun.

  PROCEDURE [XDCL] translate_message ( {
      VAR message: buf_ptr; { message to be translated
          table: string (256); { translate table
          threshold: threshold_size); { buffer allocation threshold
```

TREE MANAGEMENT DEFINITIONS

```
{ TABLE NAME:  Tree Management Definitions
{
{ PURPOSE:
{   Define Tree Management Types.
{
{ CALL FORMAT:
{  (*callc CMDTTRE)
{  TYPE
{    cmdrestab = record
{      node_ctl: node_control,
{      tcepid: ↑cell,
{      connected: boolean,
{    recend;
{
{  VAR
{    cme_tree_ptr: [XREF] ↑root,
{    table: cmdrestab,
{    key: key_record;

  TYPE
    condition_range = - 1 .. 1;

  CONST
    left_heavy = - 1,
    right_heavy = 1,
    balanced = 0;

  TYPE
    key_type = (numeric_key@, pointer_key@, string_key@);

  TYPE
    key_record = record
      case key_kind: key_type of
      = numeric_key@ =
        numeric: integer,
      = pointer_key@ =
        pointer: ↑cell,
      = string_key@ =
        string_type: ↑string ( * ),
      casend
    recend;

  TYPE
    node = record
      balance: condition_range, { balance factor for sub-tree
      association: ↑node_control, { points to user data
      key: key_record,
      llink: ↑node, { sub-tree links
      rlink: ↑node,
    recend;

  TYPE
```

```
  node_control = record
    length: executive_extent, { length of the associated table
    dump_id: string (4), { validity check, should contain user value
  recend;

TYPE
  root = record
    num_tables: integer, { number of tables in tree
    num_nodes: integer, { total number of nodes in the tree
    dump_id: string (4), { validity check, should contain user value
    type_node: key_type, { how is tree accessed
    link: ↑node,
  recend;
```

TRIM

```
{ PROCEDURE NAME: trim
{
{ PURPOSE:
{   Trim bytes needed from back of data_descriptor.
{
{ CALL FORMAT:
{   (*callc CMXPTRI)
{   trim (size,address,message)
{
{ DESCRIPTION:
{   Trim from the back of the data_descriptor the number of
{   bytes needed--i.e.  size.  If a buffer is completely used
{   up, then release it from memory.  If the entire message is
{   less than size, then return false to let the caller know
{   there is not enough bytes to satisfy the request.  If size
{   is NULL, then nothing needs to be done-- return immediately.
{

  PROCEDURE [XDCL] trim ( {
        size: non_empty_message_size; {# of bytes needed
          address: ↑cell; {where to put the bytes
      VAR msg: buf_ptr;
        threshold: threshold_size);
```

VALIDATE SECTION ADDRESS

```
{ PROCEDURE NAME: validate_section_address
{
{ PURPOSE:
{   Translate a given address into a module name and a section
{ address with  offset.
{
{ CALL FORMAT:
{   *callc dlxvsa
{   validate_section_address(address, valid_section, module_name,
{                            section_address, offset);
{
{ DESCRIPTION:
{   The module header linked list is searched, checking the section
{ address bounds for a range that contains the given address.  If
{ such a range is found and the address is valid for the MPB, the
{ boolean parameter  valid_section  is set to true, otherwise false
{ is returned.
{


  PROCEDURE [XDCL] validate_section_address
    ({
        address: ↑cell;
     VAR valid_section: boolean;
     VAR module_name: pmt$program_name;
     VAR section_address: ↑dlt$section;
     VAR offset: dlt$section_offset);
```

VECTOR TABLE USAGE DURING DCNS OPERATION

```
{ TABLE NAME:     VECTOR TABLE USAGE DURING DCNS OPERATION
{                 ---------------------------------------
{
{ DECK NAME:      CMDVECT
{
{ Vector
{ ------
{   0             Reset:  Initial System Stack Pointer.  Label RESETSP
{   1             Reset:  Initial PC
{   2             Bus Error
{   3             Address Error
{   4             Illegal Instruction
{   5             Zero Division
{   6             Check Instruction
{   7             Trap V Instruction
{   8             Privilege Violation
{   9             Trace
{  10             Line 1010 Emulator.  Unimplemented op code
{  11             Line 1111 Emulator.  Unimplemented op code
{  12-23          Reserved for future enhancements by Motorola
{  24             Spurious.  For when the interrupt cycle has been started but
{                   cleared before completion
{  25             Level 1 Interrupt Autovector.  Reserved for possible use on
{                   the 68000 Extension Bus.
{  26             Level 2 Interrupt Autovector.  Real Time Clock Interrupt
{  27             Level 3 Interrupt Autovector.  Software Timers and Clocks and
{                   Attention Switch
{  28             Level 4 Interrupt Autovector.  ISB Interrupts (scanned)
{                   8 cards (Control Bus Vector)
{  29             Level 5 Interrupt Autovector.  Extension Bus
{  30             Level 6 Interrupt Autovector.  SSC (Serial Port)
{  31             Level 7 Interrupt Autovector.  Errors.  Level 7 interrupts are
{                   non-maskable.  "ACLOW" will indicate potential power failure,
{                   cause status to be saved, and then stop.  "ERRORS" will
{                   include over-temperature condition.
{  32             TRAP 0 :  fast_bg (also called maybe_bg) (background)
{  33             TRAP 1 :  sure_bg
{  34             TRAP 2 :  fast_fg (foreground)
{  35             TRAP 3 :  fire in.  Saves registers.  Controlled recovery point.
{                   If another vector is invoked then TRAP 3 sets up firewall.
{  36             TRAP 4 :  fire out.  Resets firewall.  If no task to
{                   preempt then it restores registers and returns from exception.
{  37             TRAP 5 :  set_interval
{  38             TRAP 6 :  set_slice
{  39             TRAP 7 :  reserved for executive
{  40             TRAP 8 :  used by MCI
{  41             TRAP 9 :  reserved for I/O subsystem  (for cards)
{  42             TRAP A :  reserved for I/O subsystem       "
{  43             TRAP B :  reserved for I/O subsystem       "
{  44             TRAP C :  reserved for I/O subsystem       "
{  45             TRAP D :  used by DI Resident Debugger
{  46             TRAP E :  reserved for I/O subsystem       "
```

```
{  47              TRAP F :  used by DVM
{  48-63           Reserved for future enhancements by Motorola
{                    NOTE:  Currently DVM is using vectors 50-57.  PSR AC1A477 has
{                           been written to have them moved to a valid area.
{  64              time slice
{  66              time interval
{  65,67,69,71,
{  73,75,77,79     SCCVECT (used by DI Debugger)        (SCC)
{  68,70,72,74,
{  76,78           CIO     User Interrupt Vectors
{  80-127          Expansion
{  128-255         Available for major cards.  8 vectors alloted for each of
{                    16 possible card slots.
```

VISIT ALL NODES

```
{ PROCEDURE NAME: visit_all_nodes
{
{ PURPOSE:
{   Step through a B-tree one node at a time allowing the caller
{   to process information at each node via a user supplied
{   routine.
{
{ CALL FORMAT:
{  (*callc CMXPVAL)
{   visit_all_nodes (ptr,process,key,para,m);
{
{ DESCRIPTION:
{   The B-tree is recursively stepped through one node at a time
{   involking the user supplied routine at each node.  The user
{   supplied procedure has three parameters, a pointer to the
{   first associated table, a pointer to cell (user parameters
{   to be passed through to the process routine), and a boolean
{   value.  Stepping through the tree will continue until all
{   elements in the B-tree have been exhausted or the boolean
{   value returned via the user supplied procedure is FALSE.
{
{ NOTES AND CAUTIONS:
{   Users may manipulate trees using all defined routines with the
{   exceptions of PICK and SPICK.  Users may NOT delete nodes from
{   the tree while using visit_all_nodes.
{

  VAR
    more: boolean ; {more nodes to visit?

  PROCEDURE [XDCL] visit_all_nodes ( {
          ptr: ↑node; { pointer to current node
          process: ↑procedure (p: ↑cell; { pointer to user table
          key: integer; { associated node key
          para: ↑cell; { pointer to user parameters
      VAR m: boolean); { TRUE continue search/FALSE terminate search
          para: ↑cell); { pointer to user parameters
```

WAIT

```
{ PROCEDURE NAME: wait
{
{ PURPOSE:
{   Wait until Wakeup.
{
{ CALL FORMAT:
{   (*callc CMXMTSK)
{   wait;
{
{ DESCRIPTION:
{   The executing task is put to sleep until a Wakeup is received
{   for the task.  This allows a capability similar to 'Send Message'
{   where the message content is void.  Examples where it could
{   be used are places where a task wishes to wait for an interrupt
{   routine or other task accomplishes something before looking at
{   its intertask message queues again.
{
{   Refer to Executive ERS section 4.25.
{

  PROCEDURE [INLINE] wait;
```

WAKE UP,

```
{ PROCEDURE NAME: wake_up,
{                 wake_now,
{                 fg_wake_up
{
{ PURPOSE:
{   Wake up Waiting Task.
{
{ CALL FORMAT:
{   (*callc CMXMTSK)
{   wake_up (task, status);
{
{   wake_now (task, status);
{   fg_wake_up (task, status);
{
{ DESCRIPTION:
{   If the task has executed a wait() call, it is scheduled.
{   If not, a flag is set indicating that the next wait() call
{   is to be treated as a yield().
{   The following calls have these effects:
{
{   NAME:           TRAP NUMBER:    EFFECTS:
{
{   wake_up              0          the task is awakened.
{
{   wake_now             1          the task is awakened and a dispatch
{                                   cycle is forced, giving the task an
{                                   immediate opportunity to execute.
{
{   fg_wake_up           2          interrupt routine use only;
{                                   the task is awakened.
{

  PROCEDURE [INLINE] wake_up ( {
        t: task_ptr;
    VAR status: boolean);
```

YIELD

```
{ PROCEDURE NAME:  yield
{
{ PURPOSE:
{   Yield Control.
{
{ CALL FORMAT:
{   (*callc CMXMTSK)
{   yield;
{
{ DESCRIPTION:
{   The task voluntarily yields control of the machine.  If it is the
{   highest priority task in the current scheduling mix and no other
{   tasks are scheduled at the same priority, it will immediately
{   get control back; otherwise it will wait while currently
{   scheduled tasks at the same and higher priorities run.
{
{   In either event, when the task is re-entered, it will have a new
{   time slice of ¬16 milliseconds to execute in.  This becomes useful
{   in controlling the execution of tasks which must run at a high
{   priority, but have a history of incurring time slice faults
{   due to message translation/checksumming time or other time
{   consuming operations.
{
{   Refer to Executive ERS section 4.24.
{

  PROCEDURE [INLINE] yield;
```

# APPENDIX A

Alphabetical listing of Procedures along with decks containing code

| Procedure/Function/Table Name | Deck containing Code |
|---|---|
| abort_system | CSIABRT |
| abort_task | CMXMTSK |
| abs, max, min | CSIFUNC |
| append | CSIAPPE |
| ASCII character definitions | CMDASCI |
| assemble | CSIASSM |
| broadcast | CMIPBRO |
| buffer | CMDTBUF |
| build_header_in_place | CSIBLDH |
| call_after_interval | CMXMTIM |
| call_at_time | CMXMTIM |
| call_periodic | CMXMTIM |
| cancel_timer | CMXMTIM |
| change_timer_owner | CMXMTIM |
| checksum_next_module | DLMILPI |
| clear_allocate | CSMCLAL |
| clear_allocate_conditional | |
| clear_memory | CSMCLAL |
| clear_write_protect | CMICWP |
| close_internet_sap | RMMMSAP |
| close_status_sap | SDMSSAR |
| close_3a_sap | A3MGENE |
| clp_convert_integer_to_string | CLMI2S |
| clp_convert_string_to_integer | CLMI2S |
| clp_convert_to_rjstring | CLMI2S |
| clp_get_parameter | CLMPAR |
| clp_get_param_list | CLMPAR |
| clp_get_set_count | CLMPAR |
| clp_get_value | CLMPAR |
| clp_get_value_count | CLMPAR |
| clp_parse_command | CLMPAR |
| clp_parse_terminate | CLMSPL |
| clp_process_command | MEMCMD |
| clp_scan_parameter_list | CLMPAR |
| clp_test_parameter | CLMPAR |
| clp_test_Range | CLMPAR |
| clp_trimmed_string_size | CLMPAR |
| convert_integer_to_pointer | CSICITP |
| convert_pointer_to_integer | CSICPTI |

| Procedure/Function/Table Name | Deck containing Code |
|---|---|
| copy | CSICOPY |
| data_request_3a | A3MGENE |
| data_3a_request | B3MINET |
| dead_stop | CSIDEAS |
| decrement_module_use_count | DLMILPI |
| delay_processing | CSIDELA |
| dir_abort | DRMDIR |
| dir_change | DRMDIR |
| dir_create | DRMDIR |
| dir_delete | DRMDIR |
| dir_purge | DRMDIR |
| dir_translate | DRMDIR |
| dir_translate_and_wait | DRMDIR |
| dir_wait | DRMDIR |
| di_debug | DLMDBUG |
| di_debug_init | DLMDBUG |
| dump_close | SIMCSA |
| dump_write | SIMCSA |
| executive_error_table | CMCERTB |
| fg_trim | CSIFTRM |
| field_size | CEMGDF |
| file_access | FAMDFA |
| find | CSIFIND |
| find_first | CSIFFRS |
| find_free_node | CSIFFRE |
| find_next | CSIFNXT |
| first_byte_address | CMXPFBA |
| first_node | CSIFIRS |
| fragment | CSIFRAG |
| generic transport interface definitions | TRDGT |
| gen_data_field | CEMGDF |
| gen_template_id | CEMGDF |
| get_card_type_and_address | SDMGCTA |
| get_command_line | FAMGCL |
| get_data_field | CEMGDF |
| get_data_line | FAMGDL |
| get_express | CMXMTSK |
| get_first_byte | CMXPGFB |
| get_last_byte | CMIGLB |
| get_long_buffers | CMCBUFF |

| Procedure/Function/Table Name | Deck containing Code |
|---|---|
| get_memory | CMCBUFF |
| get_message_length | CMXPGML |
| get_mpb_extent | CMCBUFF |
| get_msg | CMXMTSK |
| get_next_status_sap | SDMSSAR |
| get_pmm_extent | CMCBUFF |
| get_short_buffers | CMCBUFF |
| get_size_n_addr | SIMGSIZ |
| get_source_address | MEMCMD |
| get_status_record | SDMGPSR |
| get_status_sap | SDMSSAR |
| grow | CSIGROW |
| increment_module_use_count | DLMILPI |
| init_root | CMIPINT |
| intertask message workcode definitions | CMDITM |
| i_compare | INMINT |
| i_compare_collated | INMINT |
| i_scan | INMINT |
| i_translate | INMINT |
| load_abs_module_and_delay | DLMILPI |
| load_abs_module_and_proceed | DLMILPI |
| load_cmd_processor_and_delay | DLMILPI |
| load_cmd_processor_and_proceed | DLMILPI |
| load_entry_point_and_delay | DLMILPI |
| load_entry_point_and_proceed | DLMILPI |
| lock_semaphore | CMXMTSK |
| log_message_enabled | LSMLSA |
| log_request | LSMLOGR |
| maybe_task | CMXMTSK |
| mdu_to_ascii | MEMM2A |
| memory owner identification definitions | CMDMOWN |
| message_dequeue | CSIQUEU |
| message_enqueue | CSIQUEU |
| modify_write_protect_byte | CSIMWPM |
| modify_write_protect_long_word | CSIMWPM |
| modify_write_protect_short_word | CSIMWPM |
| mpb_ram_template | SIDRAM |
| m_release | CMIPMLR |
| name_match | CSINAMM |
| new_interrupt | CMXMTSK |
| new_priority | CMXMTSK |
| noprempt | EXDMAC1 |
| okprempt | EXDMAC1 |

| Procedure/Function/Table Name | Deck containing Code |
|---|---|
| open_internet_sap | RMMMSAP |
| open_status_sap | SDMSSR |
| open_3a_sap | A3MGENE |
| osv_lower_to_upper | OSXTL2U |
| osv_upper_to_lower | OSXTU2L |
| pcopy | CSICOPY |
| pick | CSIPICK |
| pmp_get_date | PMMGDAT |
| pmp_get_time | PMMGDAT |
| pool_buffers | CMXMPOO |
| prefix | CSIPREF |
| put_status_record | SDMGPSR |
| read_bcd_clock | CMXMTIM |
| read_clock | CMXMTIM |
| release_message | CMCBUFF |
| request_diagnostic_entry | DGMAHWD |
| reset codes for the di | SIDRC |
| reset_di | SIMCSA |
| reset_recovery_procedure | CMISISA |
| restore_task | CMXMTSK |
| send_express | CMXMTSK |
| send_normal | CMXMTSK |
| set_bcd_clock | CMXMTIM |
| set_buffer_chain_owner | CSMCAR |
| set_memory_owner | CSISMO |
| set_recovery_procedure | CMISISA |
| set_test_lights | DGMDCR |
| set_write_protect | CMISWP |
| sfind | CSISFIN |
| sfind_first | CSISFFR |
| sfind_next | CSISFNX |
| sfind_wild_cards | CSIWILD |
| sgrow | CSISGRO |
| signal1/acquire1 | CMXMTSK |
| spick | CSISPIK |
| start_dump | SIMCSA |

| Procedure/Function/Table Name | Deck containing Code |
|---|---|
| start_named_task_and_delay | DLMILPI |
| start_named_task_and_proceed | DLMILPI |
| start_system_task | SIMCSA |
| start_task | CMXMTSK |
| stop_task | CMXMTSK |
| strip | CSISTRI |
| strip_in_place | CSISTIP |
| subfield | CSISSUB |
| suspend | CMXMTSK |
| system_configuration_table | CMCCNFG |
| task_control_block | CMDTTSK |
| thresholds | CMDTHRH |
| time | CMXMTIM |
| timer_entries | CMDTTIM |
| translate_message | CSITRAN |
| tree management definitions | CMDTTRE |
| trim | CSITRIM |
| validate_section_address | ILMILPI |
| vector table usage during dcns operation | CMDVECT |
| visit_all_nodes | CSIVIAN |
| wait | CMXMTSK |
| wake_up | CMXMTSK |
| yield | CMXMTSK |

# APPENDIX B

Alphabetical listing of types and constants referenced by the Handbook.

```
. access_status_type = (sap_opened, sap_not_opened)                    [CMDSSED]

. card_info_record = record                                            [SDDCIRD]
    card_type: hardware_resource_type,
    primary_address: integer,
    secondary_address: integer,
  recend

. clc$max_parameters = 255                                             [CLDPMAX]

. clc$max_parameter_names = 255                                        [CLDPMAX]

. clc$max_parameter_values = 255                                       [CLDPVT]

. clc$max_value_sets = 255                                             [CLDPMAX]

. clc$max_values_per_set = 255                                         [CLDPMAC]

. close_internet_sap_status = ( {                                      [B3DCSAP]
    close_sap_successful, { SAP was closed successfully
    sap_already_closed,   { Attempting to close already closed SAP
    mismatch_userid)      { Input user_id doesn't match SAP table entry

. close_3a_sap_proc_type = ↑procedure ( {                              [A3DPRCS]
      sap: intranet_sap_type;
  VAR close_status: 13a_status_type)

. clt$boolean = record                                                 [CLDBOOL]
    value: boolean,
    kind: clt$boolean_kinds,
  recend

. clt$boolean_kinds = (clc$true_false_boolean,                         [CLDBOOL]
    clc$yes_no_boolean, clc$on_off_boolean)

. clt$ccode = record                                                   [CLDCCOD]
    value: 0 .. 0ff(16),
    kind: clt$ccode_kinds,
    str: string(3),
  recend

. clt$how_parameter_given = (clc$omitted_parameter,                    [CLDPVT]
    clc$defaulted_parameter, clc$actual_parameter)

. clt$integer = record                                                 [CLDINT]
    value: integer,
    radix: 2 .. 16,
    radix_specified: boolean,
  recend
```

Appendix B: Alphabetical listing of types and constants continued

```
. clt$lexical_kinds = (clc$unknown_token, clc$space_token,          [CLDLEX]
    clc$eol_token, clc$dot_token, clc$semicolon_token,
    clc$colon_token, clc$lparen_token, clc$lbracket_token,
    clc$lbrace_token, clc$rparen_token, clc$rbracket_token,
    clc$rbrace_token, clc$uparrow_token, clc$rslant_token,
    clc$query_token, clc$comma_token, clc$ellipsis_token,
    clc$exp_token, clc$add_token, clc$sub_token, clc$mult_token,
    clc$div_token, clc$cat_token, clc$gt_token, clc$ge_token,
    clc$lt_token, clc$le_token, clc$eq_token, clc$ne_token,
    clc$string_token, clc$name_token, clc$integer_token, clc$ccode_token)

. clt$low_or_high = (clc$low, clc$high)                             [CLDPMAX]

. clt$name = record                                                 [CLDNAME]
    size: ost$name_size,
    value: ost$name,
  recend

. clt$parameter_descriptor = record                                 [CLDPDT]
    required_or_optional: clt$required_or_optional,
    min_value_sets: 1 .. clc$max_value_sets,
    max_value_sets: 1 .. clc$max_value_sets,
    min_values_per_set: 1 .. clc$max_values_per_set,
    max_values_per_set: 1 .. clc$max_values_per_set,
    value_range_allowed: (clc$value_range_not_allowed,
      clc$value_range_allowed),
    value_kind_specifier: clt$value_kind_specifier,
  recend

. clt$parameter_descriptor_table = record                           [CLDPDT]
    names: ↑array [1 .. * ] of clt$parameter_name_descriptor,
    parameters: ↑array [1 .. * ] of clt$parameter_descriptor,
  recend

. clt$parameter_name_descriptor = record                            [CLDPDT]
    name: ost$name,
    number: 1 .. clc$max_parameters,
  recend

. clt$parameter_value_table = record                                [CLDPVT]
    case built: boolean of
    = TRUE =
      parameter_list: ↑string ( * ),
      names: ↑clt$pvt_names,
      parameters: ↑clt$pvt_parameters,
      values_area: ↑clt$pvt_values_area,
      values: ↑clt$pvt_values,
    casend,
  recend

. clt$pvt_name = clt$parameter_name_descriptor                      [CLDPVT]

. clt$pvt_names = array [1 .. * ] of clt$pvt_name                   [CLDPVT]
```

```
. clt$pvt_parameter = record                                    [CLDPVT]
    how_given: clt$how_parameter_given,
    case value_set_count: 0 .. clc$max_value_sets of
    = 1 .. clc$max_value_sets =
      first_value_index: 1 .. clc$max_parameter_values,
      last_value_index: 1 .. clc$max_parameter_values,
      value_list_index: ost$string_index,
      value_list_size: ost$string_size,
      name_index: 0 .. clc$max_parameter_names,
    casend,
  recend

. clt$pvt_parameters = array [1 .. * ] of clt$pvt_parameter    [CLDPVT]

. clt$pvt_value = record                                       [CLDPVT]
    value_set_number: 1 .. clc$max_value_sets,
    value_number: 1 .. clc$max_values_per_set,
    low_or_high: clt$low_or_high,
    value: clt$value,
  recend,

. clt$pvt_values = array [1 .. * ] of clt$pvt_value            [CLDPVT]

. clt$pvt_values_area = SEQ ( * )                              [CLDPVT]

. clt$required_or_optional = record                            [CLDREQ]
    case selector: (clc$required, clc$optional, clc$optional_with_default) of
    = clc$required =
      ,
    = clc$optional =
      ,
    = clc$optional_with_default =
      default: ↑string ( * ),
    casend,
  recend

. clt$status = record                                          [CLDSTAT]
    normal: boolean,
    response_id: min_response_message_id .. max_response_message_id,
    condition: buf_ptr, { management data unit syntax }
  recend
```

```
. clt$value = record                                              [CLDVAL]
    descriptor: string (osc$max_name_size),
    case kind: clc$unknown_value .. clc$ccode_value of
    = clc$unknown_value =
      ,
    = clc$string_value =
      str: ↑ost$string_value,
    = clc$name_value =
      name: clt$name,
    = clc$integer_value =
      int: clt$integer,
    = clc$boolean_value =
      bool: clt$boolean,
    = clc$ccode_value =
      ccode: clt$ccode,
    casend,
  recend

. clt$value_kind_specifier = record                               [CLDVKS]
    keyword_values: ↑array [1 .. * ] of ost$name,
    case kind: clt$value_kinds of
    = clc$keyword_value, clc$any_value =
      ,
    = clc$name_value =
      min_name_size: ost$name_size,
      max_name_size: ost$name_size,
    = clc$string_value =
      min_string_size: ost$string_size,
      max_string_size: ost$string_size,
    = clc$integer_value =
      min_integer_value: integer,
      max_integer_value: integer,
    = clc$boolean_value =
      ,
    = clc$ccode_value =
      ,
    casend,
  recend

. clt$value_kinds = (clc$unknown_value, clc$name_value,           [CLDVLK]
    clc$string_value, clc$integer_value, clc$boolean_value,
    clc$any_value, clc$ccode_value)

. cme$max_template_id = 65535                                     [CMETMPR]

. cme$min_template_id = 0                                         [CMETMPR]

. component_status_type = record                                  [SDDCSR]
    name: string (maximum_device_name_size), { Hardware physical device name
    state: device_state_type, { device state
    status: device_status_type, { device status
  recend
```

Appendix B: Alphabetical listing of types and constants continued

```
. control_bytes = packed record                                    [B3DCOBY]
    hop_count: 0 .. 0ff(16), { Initialize to 0 and incremented
    packet_kind: packet_type, { 3B PDU data field protocol type
  recend

. data_request_3a_proc_type = ↑procedure ( {                       [A3DPRCS]
      network_id: network_id_type;
      destination_address: system_id_type;
      sap: intranet_sap_type;
  VAR data_ptr: buf_ptr;
  VAR request_processed: boolean)

. dbc$single_line = 79                                             [DBDDMP]

. destination_3b_sap_if = ↑procedure ( {                           [B3DSAPI]
        ind_params: ↑internet_ind_if)

. device_state_type = (device_on, device_off, device_down)         [SDDCSR]

. device_status_type = {                                           [SDDCSR]
   (device_not_cnfg, {
    device_cnfg, {
    device_enabled, {
    device_active)

. dlc$default_immediate_control = FALSE                            [DLDATTR]

. dlc$default_preemptibility = FALSE                               [DLDATTR]

. dlc$default_priority = 0                                         [DLDATTR]

. dlc$max_section_checksum = 0ffff(16)                             [DLDCCHK]

. dlc$max_section_length = dlc$max_section_offset                  [DLDCSCA]

. dlc$max_section_offset = 7fffffff(16)                            [DLDCSCA]

. dlc$max_section_ordinal = 0ffff(16)                              [DLDCSCA]

. dlc$maximum_68000_address = 7fffffff(16)                         [DLD68AD]

. dlt$ampm_time = string (8)                                       [DLDTIME]

. dlt$checksum = 0 .. dlc$max_section_checksum                     [DLDCCHK]
```

Appendix B: Alphabetical listing of types and constants continued

```
. dlt$date = packed record                                          [DLDDATE]
    fill: 0 ..1f(16),
    case date_format: dlt$date_formats of
    = dlc$month_date =
      month: dlt$month_date, { month DD, YYYY }
    = dlc$mdy_date =
      mdy: dlt$mdy_date, { MM/DD/YY }
    = dlc$iso_date =
      iso: dlt$iso_date, { YYYY-MM-DD }
    = dlc$ordinal_date =
      ordinal: dlt$ordinal_date, { YYYYDDD }
    = dlc$dmy_date =
      dmy: dlt$dmy_date { DD/MM/YY }
    casend,
  recend

. dlt$date_formats = (dlc$default_date, dlc$month_date,              [DLDDATE]
    dlc$mdy_date, dlc$iso_date, dlc$ordinal_date, dlc$dmy_date)

. dlt$dmy_date = string (8)                                         [DLDDATE]

. dlt$entry_description = record                                    [DLDCEPT]
    node: node_control,
    name: pmt$program_name,
    address: dlt$68000_address,
    module_header_address: ↑dlt$module_header,
    link_address: ↑dlt$entry_description,
    declaration_matching_required: boolean,
    declaration_matching_value: string (8),
    language: dlt$module_generator,
  recend

. dlt$hms_time = string (8)                                         [DLDTIME]

. dlt$iso_date = string (10)                                        [DLDDATE]

. dlt$load_id_ptr = dlt$module_header_ptr                           [DLDLPTR]

. dlt$maximum_modules = 0 .. dlc$max_section_ordinal                [DLDCMM]

. dlt$mdy_date = string (8)                                         [DLDDATE]

. dlt$millisecond_time = string (12)                                [DLDTIME]

. dlt$module_attributes = set of                                    [DLDCMA]
   (dlc$nonbindable, dlc$nonexecutable);

. dlt$module_generator = (dlc$algol, dlc$apl, dlc$basic,            [DLDCMG]
    dlc$cobol, dlc$assembler, dlc$fortran,
    dlc$object_library_generator, dlc$pascal, dlc$cybil,
    dlc$pl_i, dlc$unknown_generator, dlc$the_c_language, dlc$ada)
```

```
. dlt$module_header = record                                    [DLDCMHP]
    link_address: dlt$module_header_ptr,
    mod_head: dlt$module_identification,
    allocated_sections: array [0 .. * ] of dlt$section_identification,
  recend

. dlt$module_header_ptr = ↑dlt$module_header                    [DLDCMDP]

. dlt$module_identification = record                            [DLDCMHD]
    name: pmt$program_name,
    kind: dlt$module_kind,
    time_created: dlt$time,
    date_created: dlt$date,
    attributes: dlt$module_attributes,
    breakpoint_set: boolean,
    retain: boolean,
    member_of_internal_set: boolean,
    use_count: dlt$maximum_modules,
    reference_list: ↑dlt$module_reference,
    module_status: dlt$module_status,
    entry_address: ↑dlt$entry_description,
    greatest_section_ordinal: dlt$section_ordinal,
    transfer_symbol_address: ↑dlt$entry_description,
  recend

. dlt$module_kind = (dlc$mi_virtual_state,                      [DLDCMK]
    dlc$vector_virtual_state, dlc$iou, dlc$motorola_68000,
    dlc$p_code, dlc$motorola_68000_absolute);

. dlt$module_reference = record                                 [DLDCMR]
    link_address: ↑dlt$module_reference,
    reference_link: ↑dlt$module_header,
  recend

. dlt$module_status =                                           [DLDCMS]
   (dlc$active, dlc$deloaded, dlc$load_in_progress)

. dlt$month_date = string (18)                                  [DLDDATE]

. dlt$ordinal_date = string (7)                                 [DLDDATE]

. dlt$section = array [1 .. *] of 0 .. 255                      [DLDCSIR]

. dlt$section_access_attribute = (dlc$read, dlc$write,          [DLDCSAA]
    dlc$execute, dlc$binding, dlc$read_other, dlc$write_other,
    dlc$execute_other, dlc$binding_other);

. dlt$section_access_attributes = set of                        [DLDCSAA]
    dlt$section_access_attribute
```

```
. dlt$section_identification = record                              [DLDCSIR]
    checksum: dlt$checksum,
    length: dlt$section_length,
    attributes: dlt$section_access_attributes,

    case 1  .. 2 OF

    = 1 =
      address: ↑dlt$68000_absolute,
      module_kind: dlt$module_kind,

    = 2 =
      section_address: ↑dlt$section,
      kind: dlt$section_kind,
    casend,
  recend

. dlt$section_kind = (dlc$code_section, dlc$binding_section,       [DLDCSK]
    dlc$working_storage_section, dlc$common_block,
    dlc$extensible_working_storage, dlc$extensible_common_block,
    dlc$line_table_section)

. dlt$section_length = 0 .. dlc$max_section_length                 [DLDCSCA]

. dlt$section_offset = 0 .. dlc$max_section_offset                 [DLDCSCA]

. dlt$section_ordinal = 0 .. dlc$max_section_ordinal               [DLDCSCA]

. dlt$time = packed record                                         [DLDTIME]
    fill: 0 .. 3f(16),
    case time_format: dlt$time_formats of
    = dlc$ampm_time =
      ampm: dlt$ampm_time, { HH:MM AM or PM }
    = dlc$hms_time =
      hms: dlt$hms_time, { HH:MM:SS }
    = dlc$millisecond_time =
      millisecond: dlt$millisecond_time, { HH:MM:SS.MMM }
    casend,
  recend

. dlt$time_formats = (dlc$default_time, dlc$ampm_time,              [DLDTIME]
    dlc$hms_time, dlc$millisecond_time)

. dlt$68000_address = 0 .. dlc$maximum_68000_address               [DLD68AD]

. file_access_mode = (read_write, write_only, read_only)           [CMDFAME]

. file_access_name = string ( * <= max_file_name_len)              [CMDFAME]

. file_access_title = string ( * <= max_title_name_len)            [CMDFAME]

. file_access_type = (sequential, random)                          [CMDFAME]
```

```
. file_control = record                                          [CMDFAME]
    { required of user for each request }
    request_code: file_requests,
    response_procedure: ↑procedure (a: ↑file_control), { procedure to call
    { when returning the file access response, if NIL control
    { will not be returned until the request is complete

    { returned by DFA }
    fcb: ↑cell, { internal file control block returned on initial request
    access_complete: boolean,
    response_code: file_responses,
    reject_code: file_reject,

    { required for request_code = create_file, open_file, delete_file }
    title_name: ↑file_access_title,
    file_name: ↑file_access_name,

    { required for request_code = create_file, open_file }
    access_mode: file_access_mode,
    access_type: file_access_type,

    { required for request_code = read_file }
    read_length: read_length, { byte count of data to be read

    { required for request_code = write_file }
    data_buffer: buf_ptr, { appended to by DFA on read_file

    { required for request_code = seek_file }
    origin: file_origin,
    offset: file_offset, { bytes from origin

    { optional }
    user_id: ↑cell, { User identifier
    quality: service_quality, { not currently used

    { returned by DFA }
    current_position: file_position, { bytes from BOI
    file_length: file_size, { the length of the file in bytes
    line_number: line_number, { updated by get_command_line, get_data_line
    file_server: gt_sap, { file server transport address
  recend

. file_offset = integer                                          [CMDFAME]

. file_origin = {                                                [CMDFAME]
    (beginning_of_file, {
     current_position,{
     end_of_file)

. file_position = 0 .. max_byte_file_size                        [CMDFAME]
```

```
. file_reject = (                    {                                  [CMDFAME]
    unspecified_error,               { defined by CDNA GDS
    security_error,                  { "
    insufficient_space,              { "
    i_o_error,                       { "
    file_does_not_exist,             { "
    invalid_file_position,           { "
    file_service_unavailable,        { "
    protocol_error,                  { "
    unexpected_file_close,           { "
    no_seek_on_sequential_file,      { "
    bad_byte_count,                  { read/write length = 0
    bad_file_name,                   { too many or garbage characters
    beyond_end_of_file,              { read attempt past EOI
    fcb_active,                      { FCB already active
    illformatted_request,            { bad request_code or file_origin
    purge_busy,                      { purge already in progress
    unknown_fcb,                     { invalid FCB on DFA service request
    usage_conflict)                  { conflict with another user

. file_requests = ( {                                                  [CMDFAME]
    create_file, {
    open_file, {
    delete_file, {
    close_file, {
    write_file, {
    read_file, {
    seek_file)

. file_responses = (request_confirmed, request_rejected)               [CMDFAME]

. file_size = 0 .. max_byte_file_size                                  [CMDFAME]

. force_close_if = ↑procedure (                                        [B3DFCIS]
        sap_id: sap_id_type;
        user_id: ↑cell)

. four_byte_statistic_record = record                                  [SMDSTAT]
    header: hdr_type,
    data: integer,
  recend

. generic_sap = internet_address                                       [TRDSAP]

. hardware_resource_type = (mpb, cim, esci, reserved_3,                [SDDSSTD]
    reserved_4, reserved_5, reserved_6, pim, pmm, smm,
    reserved_10, reserved_11, disc, mci, dci, slot_empty,
    lim, port, bank)
```

```
. hdr_type = packed record { header field                          [METMDU]
    reserved: boolean, { reserved bit
    data_element_type: mdu_field_type, { 4 bit type field
    data_compress_flg, { compress ASCII
    field_flg, { true if end of field
    command_flg: boolean, { true if format command
    length: 0 .. 255, { length of data field
  recend

. internet_address = record                                        [B3DINAD]
    system_addr: system_address,
    sap_id: sap_id_type,
  recend

. internet_ind_if = record                                         [B3DSAPI]
    multicast: boolean, { INPUT - TRUE=multicast, FALSE=datagram
    checksum: boolean, { INPUT - TRUE if message was checksummed
    source_address: internet_address, { INPUT
    destination_address: internet_address, { INPUT
    control: control_bytes, { INPUT - hop_count and packet_type
    user_id: ↑cell, { INPUT - user ID for this SAP entry
    data: buf_ptr, { INPUT - message buffer descriptor address
  recend

. internet_req_if = record {.Internet request parameters           [B3DREQP]
    source_address: internet_address, { INPUT
    destination_address: internet_address, { INPUT
    packet_kind: packet_type, { INPUT - user protocol
    checksum: boolean, { INPUT - TRUE selects checksumming
    data: buf_ptr, { INPUT - message to be sent
  recend

. internet_request_address = ↑procedure ( {                        [B3DREQP]
       req_param: ↑internet_req_if;
    VAR return_code: internet_return_codes)

. internet_return_codes = ( {                                      [B3DRTNT]
    internet_success, { No internet error
    ineterror_nil_param, { NIL param ptr supplied
    ineterror_sosap, { illegal source SAP (not in SAP table range
    ineterror_dssap, { illegal desination SAP (not in SAP table range
    ineterror_data)  { no data or too much data

. intranet_sap_type   = 0 .. 65535                                 [A3DHDRS]

. intranet_sds_expanded_data = record                              [A3DSTS]
   messages_transmitted: four_byte_statistic_record,
   messages_received: four_byte_statistic_record,
   broadcast_messages_received: four_byte_statistic_record,
   messages_discarded: four_byte_statistic_record,
   congested_state_count: two_byte_statistic_record,
   un_congested_state_count: two_byte_statistic_record,
   other_state_count: two_byte_statistic_record,
   congested_state_transition : two_byte_statistic_record,
  recend
```

```
. line_number = 0 .. 0ffff(16)                                    [CMDFAME]

. log_msg_id_type = min_log_message_id .. max_log_message_id      [CMEECCR]

. log_priority = (log_critical, log_high, log_medium, log_low)    [LSDALDS]

. 13a_status_type = (request_processed,                           [A3DPROT]
                     sap_out_of_range,
                     sap_active,
                     sap_not_active)

. max_byte_file_size = 07fffffff(16)                              [CMDFAME]

. max_data_length = 1470                                          [B3DDFAU]

. max_file_name_len = 63                                          [CMDFAME]

. max_log_message_id = 32999                                      [CMEECCR]

. max_response_message_id = 65535                                 [CMEECCR]

. maximum_device_name_size = 11                                   [SDDCSR]

. mdu_field_size = 32000                                          [METMDU]

. mdu_field_type = bin_str .. format                              [METMDU]
  where:
     bin_str = 0,
     bin_octet = 1,
     char_octet = 2,
     bin_int = 3,
     bin_sint = 4,
     bcd_char = 7,
     format = 8

. min_log_message_id = 0                                          [CMEECCR]

. min_response_message_id = 33000                                 [CMEECCR]

. network_id_type = integer                                       [B3DINAD]

. network_range_type = (                                          [CMDNIB]
  hdlc_network,
  esci_network,
  mci_network,
  x25_network
  )

. network_status_type = (                                         [CMDNIB]
  net_up,
  net_inactive,
  net_congested,
  net_terminate
  )
```

Appendix B: Alphabetical 1sting of types and constants continued

```
. nib_type = record                                                    [CMDNIB]
    next_nib:        ↑nib_type,              { chain to next nib
    network_type:    network_range_type,     { network solution type
    network_status: network_status_type,     { network solution status
    network_id:      network_id_type,        { network solution id
    network_name:    c1t$name,               { network solution name
    network_cost:    0 .. 0ffff(16),         { network solution cost
    relay_allowed:         boolean,          { network allows relay
    multicast_network:     boolean,          { multicast nw indication
    cdna_routing_info_nw: boolean,           { routing info indication
    rotary:                boolean,          { hdlc rotary indication
    cdna_xerox_broadcast_addr: system_id_type, { broadcast addr for nw.
    max_data_unit_size:    0 .. 0ffff(16), { maximum data unit size
    intranet_header_size: 0 .. 0ffff(16), { 3A header size
    congestion_threshold:     0 .. 255,    { system becomes congested
    un_congestion_threshold: 0 .. 255,     { system becomes uncongested
    lib_ptr:                 ↑cell,        { chain to associated LIB
    intranet_sds_data1:     intranet_sds_expanded_data, { to collect statistics
    intranet_sds_data2:     intranet_sds_expanded_data, { to collect statistics
    intranet_sds_data_ptr: ↑intranet_sds_expanded_data, {current collection buffer
  recend

. open_internet_sap_status = ( {                                       [B3DOSAP]
    open_sap_successful, { SAP was opened successfully
    illegal_dedicated_sap, { This dedicated SAP ID not in expected range
    nil_parameter_pointer, { NIL provided as input or output parameter ptr
    no_destination_proc, { NIL provided as 3B data destination procedure
    sap_already_opened, { This dedicated SAP is already open
    no_sap_entries_available, { All SAP table entries are being used
    sap_3b_insuf_resorc, { Insufficient resources to create SAP entry
    internet_down)  { INTERNET not available

. open_sap_input_parameters = record                                   [B3DOSIF]
    sap_id: sap_id_type, { If <> 0: Requested Dedicated SAP ID
    user_id: ↑cell, { user identifier
    destination: destination_3b_sap_if, { Proc to receive 3B indications
    force_close: force_close_if, { Procedure for Routing M-E to close SAP
  recend

. open_sap_output_parameters = record { SAP                            [B3DOSIF]
    local_internet_address: internet_address, { w/ assigned SAP ID
    internet_request: internet_request_address,
    maximum_request_length: 1 .. max_data_length,
  recend

. osc$max_name_size = 31                                               [OSDNAME]

. osc$max_string_size = 256                                            [OSDSTRD]

. ost$ampm_time = string (8)                                           [OSDTIME]
```

```
. ost$date = record                                              [OSDDATE]
    case date_format: ost$date_formats of
    = osc$month_date =
      month: ost$month_date, { month DD, YYYY }
    = osc$mdy_date =
      mdy: ost$mdy_date, { MM/DD/YY }
    = osc$iso_date =
      iso: ost$iso_date, { YYYY-MM-DD }
    = osc$ordinal_date =
      ordinal: ost$ordinal_date, { YYYYDDD }
    = osc$dmy_date =
      dmy: ost$dmy_date { DD/MM/YY }
    casend,
  recend

. ost$date_formats =                                             [OSDDATE]
   (osc$default_date, osc$month_date,   osc$mdy_date,
    osc$iso_date,     osc$ordinal_date, osc$dmy_date)

. ost$dmy_date = string (8)                                      [OSDDATE]

. ost$hms_time = string (8)                                      [OSDTIME]

. ost$iso_date = string (10)                                     [OSDDATE]

. ost$mdy_date = string (8)                                      [OSDDATE]

. ost$millisecond_time = string (12)                             [OSDTIME]

. ost$month_date = string (18)                                   [OSDDATE]

. ost$name = string (osc$max_name_size)                          [OSDNAME]

. ost$name_size = 1 .. osc$max_name_size                         [OSDNAME]

. ost$ordinal_date = string (7)                                  [OSDDATE]

. ost$string = record                                            [OSDSTRD]
    size: ost$string_size,
    value: string (osc$max_string_size),
  recend

. ost$string_index = 1 .. osc$max_string_size + 1                [OSDSTRD]

. ost$string_size = 0 .. osc$max_string_size                     [OSDSTRD]
```

```
. ost$time = record                                                    [OSDTIME]
    case time_format: ost$time_formats of
    = osc$ampm_time =
      ampm: ost$ampm_time, { HH:MM AM or PM }
    = osc$hms_time =
      hms: ost$hms_time, { HH:MM:SS }
    = osc$millisecond_time =
      millisecond: ost$millisecond_time, { HH:MM:SS.MMM }
    casend,
  recend

. ost$time_formats =                                                   [OSDTIME]
   (osc$default_time, osc$ampm_time,
    osc$hms_time,     osc$millisecond_time)

. packet_type = 0 .. 0ff(16)                                           [B3DPCKT]
    {
    { Known values for packet_type for Internet requests and indications
    {
       CONST
         unknown_packet_type = 0,
         xerox_routing_info_packet = 1,
         xerox_echo_packet = 2,
         xerox_error_packet = 3,
         xerox_packet_exchange = 4,
         xerox_sequenced_packet = 5,
         experimental_packet = 16,
         cdna_routing_info_packet = 17,
         cdna_directory_packet = 18,
         cdna_command_packet = 19,
         cdna_log_packet = 20

. pmt$program_name = ost$name                                          [PMDNAME]

. protocol_range_type = 0 .. 0ff(16)                                   [A3DHDRS]

. read_file_status = {                                                 [CMDFAME]
   (read_ok, {
    read_eof, {
    line_too_long, {
    access_error)

. read_length = 1 .. 0ffff(16)                                         [CMDFAME]

. sap_id_type = 0 .. 0ffff(16)                                         [B3DINAD]

. sat$max_dump_size = 0 .. 4096                                        [CMDSISA]

. sat$recovery_block = record                                          [CMDSISA]
    procedure_address: ↑procedure, { pointer to code and static link address
    sa_dump_identifier: ↑cell, { sat$dump_identifier, ptr to dump control block
    previous_link: ↑sat$recovery_block, { previous recovery block on stack
  recend
```

Appendix B: Alphabetical listing of types and constants continued

```
. service_quality = 0 .. 3                                          [CMDFAME]

. software_sap_range = 1 .. 0ffff(16)                               [CMDSSED]

. system_id_type = record                                           [B3DINAD]
    upper: 0 .. 0ffff(16),
    lower: integer,
  recend

. system_status_table_type =                                        [SDDSSTD]
   (major_card_table_type, lim_table_type,
    port_table_type,       smm_bank_table_type, pmm_bank_table_type)

. task_attributes = record                                          [DLDATTR]
    stack_allocation: stack_size,
    task_priority: priorities,
    preemptable: boolean,
    immediate_control: boolean,
  recend

. template_id_type = cme$min_template_id .. cme$max_template_id

. two_byte_statistic_record = record                                [SMDSTAT]
    header: hdr_type,
    data: 0 .. 0ffff(16),
  recend

. user_datagram_proc_type = ↑procedure ( {                          [A3DPRCS]
    multicast: boolean;
    receiving_network_id: network_id_type;
    originating_system_id: system_id_type; { For ESCI or MCI networks
    VAR data_ptr: buf_ptr)

. user_status_proc_type = ↑procedure ( {                            [A3DPRCS]
    network_nib: ↑nib_type)
```